SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mia Kokić

Zagreb, 2014.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor: Student:

Prof. dr. sc. Bojan Jerbić, dipl. ing. Mia Kokić

Zagreb, 2014.

Izjavljujem da sam završni rad izradila sam samostalno, služeći se literaturom i znanjem stečenim tijekom studija.

Zahvaljujem se mentoru prof. dr. sc. Bojanu Jerbiću na pomoći i sugestijama prilikom izrade završnog rada. Također se želim zahvaliti asistentu dr. sc. Tomislavu Stipančiću na korisnim savjetima i pomoći pri izradi završnog rada.

Isto tako bih se zahvalila kolegi Tomislavu Tomašiću na suradnji oko izrade rada.

Mia Kokić

SADRŽAJ

POPIS SLIKA

SAŽETAK

Ovaj završni rad sastoji se od dva dijela. U prvom dijelu riješio se problem ponašanja robota u ovisnosti o stanju vizijskog sustava na robotu za različite modele postavljene pred kameru dok se robot istovremeno kreće po nekoj prije definiranoj putanji . Ponašanje robota isprogramirano je na način da se robot i njegova radna okolina snimaju sa dva Microsoft Kinect uređaja kako bi se mogla detektirati prepreka ukoliko se ista pojavi na putanji robota.

U drugom dijelu napravljena je implementacija spomenutog programa na klasičnoj strukturi robota zbog manjih dimenzija i lakšeg korištenja u odnosu na paralelnu strukturu na kojoj je program prvenstveno napravljen.

1. UVOD

Zadatak rada je oblikovati virtualnu okolinu robota koja odražava podatke prikupljene vizijskim sustavom. Virtualna okolina treba poslužiti za analizu stanja robota i njegovog radnog prostora u toj istoj okolini radi donošenja odluka o ponašanju.

Robot preko senzora prikuplja informacije o svojoj okolini. Kako bi robot mogao sam odlučiti o svojoj sljedećoj akciji on mora obraditi te informacije, a zadatak korisnika je isprogramirati djelovanje robota ovisno o prikupljenim informacijama. U ovom radu se koriste dva vizijska sustava kao senzori. Za pouzdano donošenje odluka potrebno je detaljnije utvrditi karakteristike objekata od interesa stoga će se koristiti 2D kamera robota za prepoznavanje oblika i donošenje odluka o ponašanju u radnoj okolini te stereo-vizijski sustav Microsoft Kinect za prikupljanje informacija o stanju okoline, ali i o stanju robota.

Program koji koristi Microsoft Kinect je implementiran na dvije strukture robota, paralelnu i klasičnu strukturu, a vizijski sustav je primijenjen samo na paralelnoj strukturi robota zbog jednostavnije implementacije.

1.1. Vizijski sustav

Ideja je bila primijeniti vizijski sustav robota paralelne strukture za prepoznavanje konkretnih događaja ili pojava u radnoj okolini robota. Korištenje Microsoft Kinect uređaja služi kako bi se snimio cijeli radni prostor i sam robot u njemu, a vizijski sustav služi kako bi se objekti od interesa prepoznali i opisali. Sam vizijski sustav sa 2D kamerom je prilagođen za detaljno prepoznavanje oblika prezentiranih kameri i lakše se može primijeniti u ovom slučaju od Microsoft Kinect uređaja. Informacije dobivene od strane Microsoft Kinect uređaja tako služe za oblikovanje šire slike dok informacije dobivene od strane vizijskog sustava služe za detaljniju analizu radne okoline robota.

Odlučivanje robota u ovisnosti o stanju senzora temelji se na obradi prikupljenih podataka iz Microsoft Kinect uređaja i vizijskog sustava na robotu na način da robot u radnom prostoru može izbjegavati prepreke i prilagođavati svoju putanju ovisno o položaju prepreke ili da može primijeniti neku drugu akciju definiranu od strane korisnika.

Kako bi robot u bilo kojem trenutku mogao reagirati s obzirom na stanje senzora njegova putanja je podijeljena na manje segmente zbog činjenice da se svaki program blokira dok robot ne izvrši gibanje koje mu je zadano. Tako ako zadamo robotu gibanje od točke A do točke B on neće moći napraviti ništa drugo dok ne izvrši zadano gibanje. Putanja je podijeljena na više segmenata od po nekoliko centimetara tako da kada vizijski sustav

detektira na primjer trokut robot se može zaustaviti i ostati tako dok se stanje senzora opet ne promijeni.

Problem sa korištenjem vizijskog sustava je u tome što se program koji ga koristi ne može implementirati kao pozadinski već ga je potrebno direktno uključiti u kod. Kod korištenja privjeska za učenje putanju je jednostavno podijeliti na segmente koristeći pozicijske registre i ofsete. Prilikom pisanja koda u Karel-u eksperimentalno se određuje diferencijal za koji će se robot pomicati prelazeći svoju putanju. Vrijednost tog diferencijala ovisi o brzini gibanja. Zbog načina na koji robot planira putanju gibanja, ako je put gibanja prekratak on će na tom putu koristiti brzinu manju od nominalne. Što je veći broj segmenata, robot može brže prilagoditi putanju ovisno o stanju radne okoline, ali se time smanjuje brzina gibanja. Zbog toga je za neku željenu brzinu potrebno odrediti optimalan broj segmenata, kako bi se ostvarila što veća brzina uz zadovoljavajuću brzinu reakcije na promjene putanje.

1.2. Implementacija na klasičnoj strukturi

U dinamičnoj okolini u kojoj robot i čovjek istovremeno surađuju na obavljanju zadatka, potrebno je poznavati položaj čovjeka u prostoru, kako bi se izbjegle eventualne ozljede, te ubrzalo izvođenje zadatka.

Za oblikovanje virtualne okoline korištena su dva Microsoft Kinect uređaja koji daju informacije o položaju čovjeka i prepreka te dubinsku mapu. Dubinska mapa služi kako bi se izračunali eventualni sudari s preprekama na putanji robota. U radu su korištena dva Microsoft Kinect uređaja čime je u nominalnim situacijama kada nema preklapanja sa robotom dodan stupanj redundancije koji dodatno osigurava detekciju prepreke.

Program je prvenstveno razvijen na robotu paralelne strukture koji iako brz zauzima veliki volumen. Sustav zanemaruje prostor koji robot zauzima pa je zbog složenosti strukture i njene izmjene u različitim položajima robota zanemaren znatno veći volumen prostora nego što bi to bilo potrebno sa klasičnom strukturom robota. Znatno točnije reprezentacija robota i njegove radne okoline dobije se implementacijom na klasičnoj strukturi. Klasična struktura robota je jednostavnija za prikaz i takav robot zauzima manji volumen i lakši je za upravljanje.

Kako bi robot u bilo kojem trenutku mogao reagirati na prepreku i promijeniti svoje gibanje, njegova putanja je podijeljena na segmente od po nekoliko dijelova. Duljina segmenta određena je eksperimentalno poznajući način na koji robot planira svoju putanju.

Slika 1 Fanuc M-10*i*A

2. VIZIJSKI SUSTAV iRVision

Vizijski sustav isproban je na Fanuc M3-iA robotu delta strukture, a opširnije o iRVision sustavu slijedi u nastavku.

Slika 2 Fanuc M3-*i*A

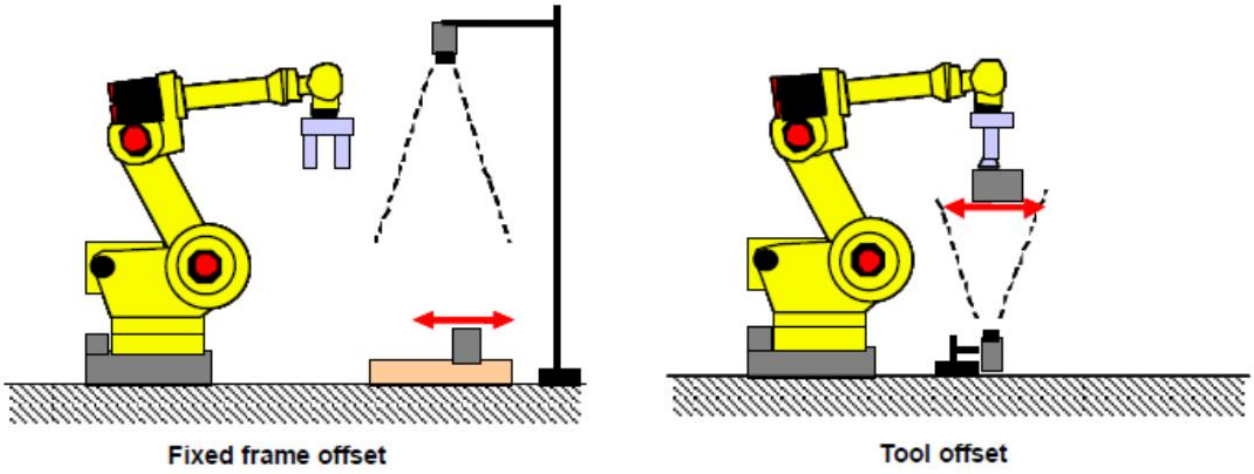
iRVision (*Integrated Robot Vision*) u prijevodu znači integrirani robotski vid. Sustav je integriran u kontroler, a vizijski sustav se u potpunosti odvija u upravljačkoj jedinici i nije potreban dodatak hardver. Upravljačkoj jedinici se pristupa računalom uz pomoć nternet preglednika upisujući IP adresu upravljačke jedinice, a vizijski procesi se pohranjuju u memoriju upravljačke jedinice. iRVision omogućuje četiri procesa :

1. Normalni proces 2D vizije
2. 2½D Vision proces depaletiziranja
3. 2D Multiview Vision proces
4. Visual Line Tracking 2D Vision proces

U ovom programu korišten je normalni proces 2D vizije koji dostavlja informaciju o X, Y, R robotskih koordinata statičnih dijelova.

Nakon što smo pristupili upravljačkoj jedinici iz iRVision sekcije odaberemo opciju *Vision Setup*. U *Select Vision Setup* sekciji uređujemo postavke kamere, vršimo kalibraciju sustava i kreiramo program za prepoznavanje objekata.

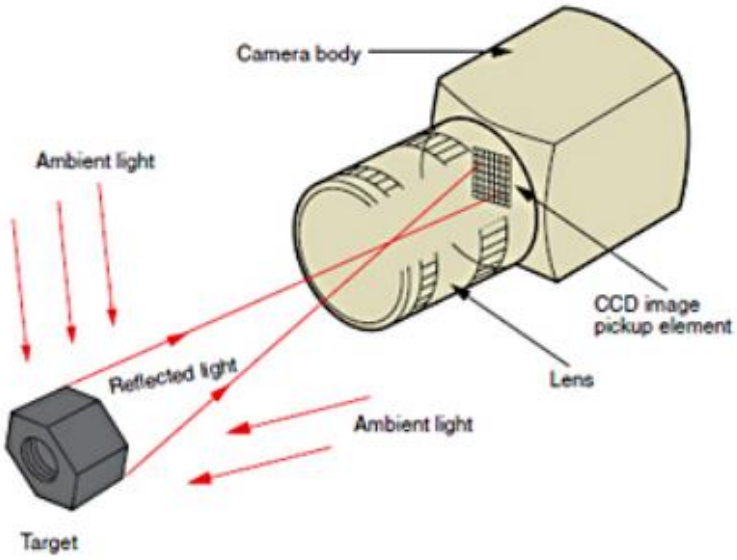
iRVision proces može koristiti dvoje vrste ofseta. *Fixed Frame* i *Tool Offset*. U ovom slučaju koristiti se prvi jer je kamera fiksna,a ne montirana na robotsku ruku kao što je slučaj kod klasične strukture. Sustav koristi Sony XC – 56 kameru i leću.



Slika 3 Offset kod fiksne i pomične kamere

Za pokretanje vizijskog sustava potrebno je zadovoljiti određene hardverske zahtjeve.

- R - 30*i*A kontroler
- Kamera i leća
- Kabel za kameru
- PC
- Ethernet kabel



Slika 4 Sony XC-56 kamera i leća

Na računalu je potrebno imati:

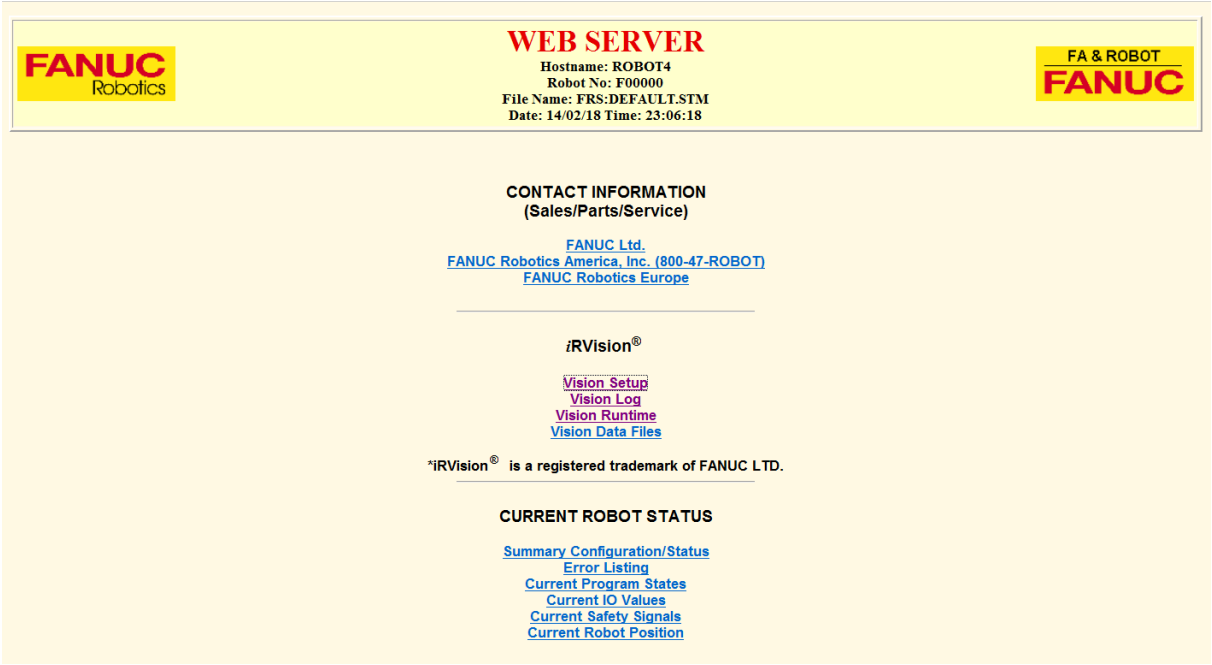
- Microsoft Windows (XP Proffesional, XP Home, Vista Business)
- Web pretraživač (Internet Explorer 6 - 9)
- Komunikacijski port (Ethernet 10/100 Base - T)

Također je potrebno postaviti IP adresu robota, PC IP adresu i instalirati UIF kontrole. Nakon upisivanja IP adrese u Internet Explorer sustav je spreman za rad.

2.1. Postavke kamere

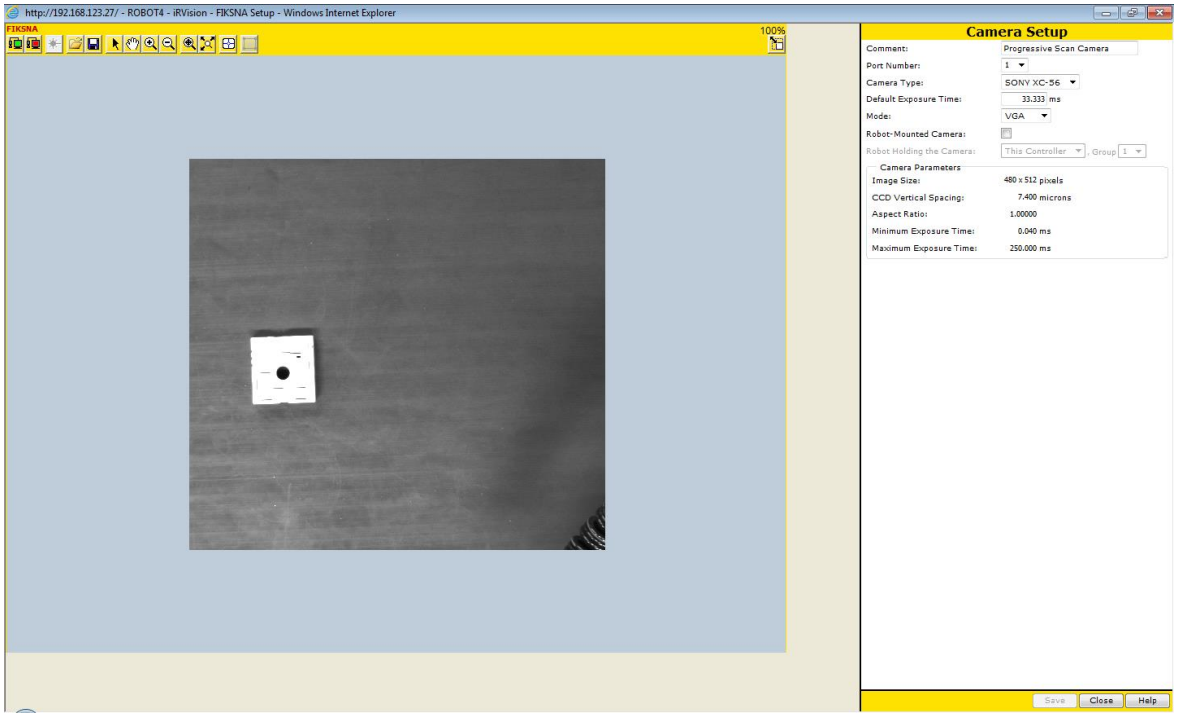
Kod postavki kamere potrebno je obratiti pozornost na nekoliko stavki ponuđenih u izborniku. To su ime kamere, broj porta, tip kamere i vrijeme ekspozicije. Parametri kamere su vidljivi unutar izbornika i na njih ne možemo utjecati.

Prije kalibracije kamere potrebno je povezati kameru s računalom. Kamera nije direktno povezana s računalom već preko upravljačke jedinice, a ona je povezana sa računalom preko IP adrese koju treba postaviti. Kada smo povezali upravljačku jedinicu sa računalom pokrenemo *Internet Explorer* i upišemo IP adresu robota. Za robot M – 3*i*A to je 192.168.123.27. Otvara se sljedeći prozor.



Slika 5 Pokretanje iRVision sustava

Prvo je potrebno postaviti kameru, a potom kalibrirati. To radimo ulazom u *Vision Setup.* Postavke fiksne kamere odabiremo klikom na *Camera Setup Tools* i kreiramo novu kameru pod nazivom FIKSNA tipa *Progressive Scan Camera*. Pod *Camera Type* odaberemo koju kameru koristimo, a to je SONY XC – 56 koja je kompatibilna sa Fanuc robotima *i*RVision sučeljem. Sljedeća stavka je *Default Exposure Time*, odnosno vrijeme ekspozicije tu ćemo postaviti 10 ms. Otvara se sljedeći prozor.

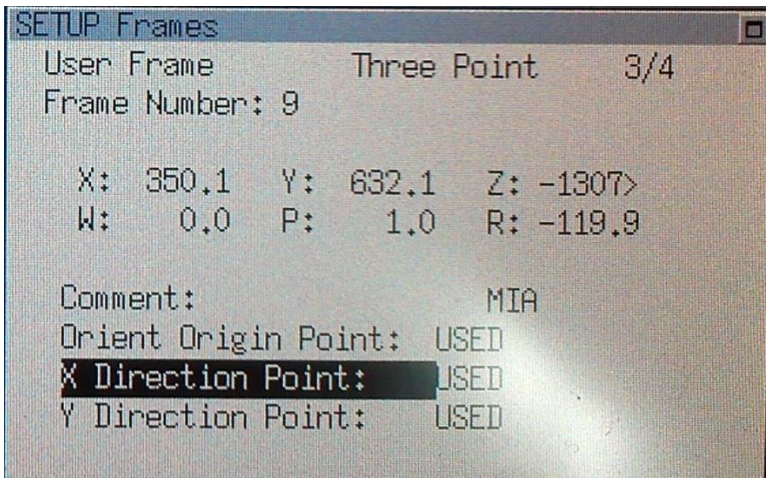


Slika 6 Postavke kamere

2.2. Kalibracija sustava

Prije svega potrebno je kalibrirati slike koju dobijemo s kamere pokrećući novi kalibracijski postupak. Sa pločom koja pripada kalibracijskoj metodi kalibriramo slike.

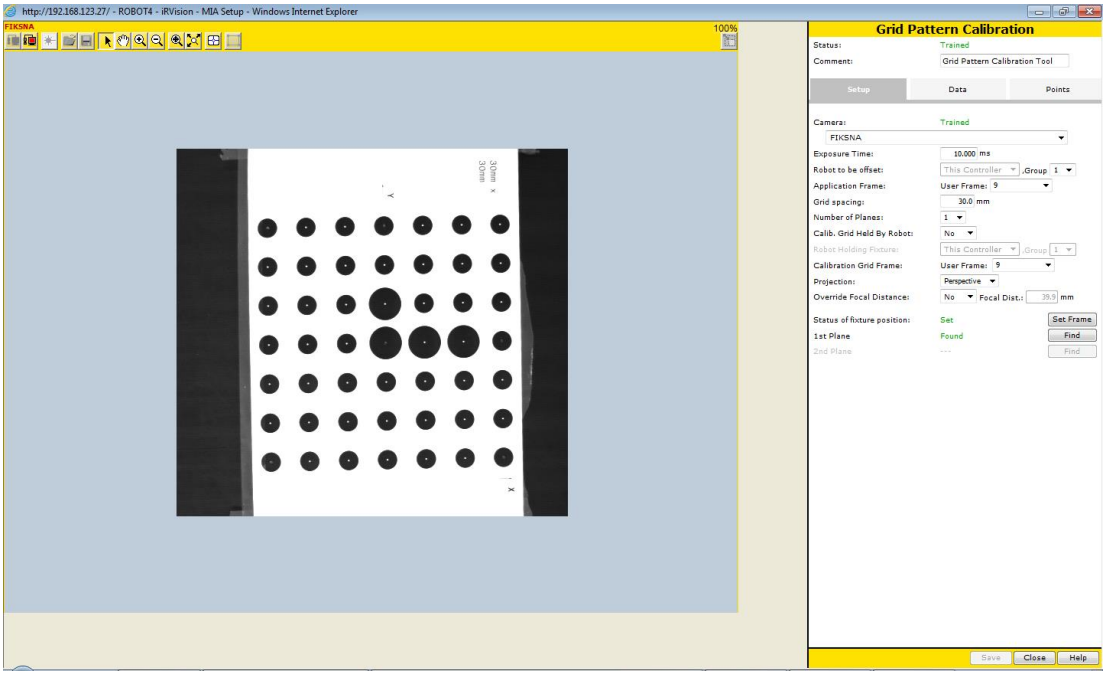
Kalibracija se vrši po principu odabira tri točke na kalibracijskoj ploči tzv. *Three - point Calibration.* Prvo je potrebno odabrati *User Frame* u kojem će se spremiti točke. U ovom radu odabran je *User Frame* 9. Nakon što smo postavili kalibracijsku ploču na traku potrebno je robota dovesti u tri točke (centralnu, u smjeru osi x i u smjeru osi y). Točke su istaknute na kalibracijskoj ploči. Kada smo dotakli sve tri točke, spremimo ih i imamo definiran korisnički koordinatni sustav koji ćemo koristiti kod kalibracije.



Slika 7 Kalibracija pomoću tri točke

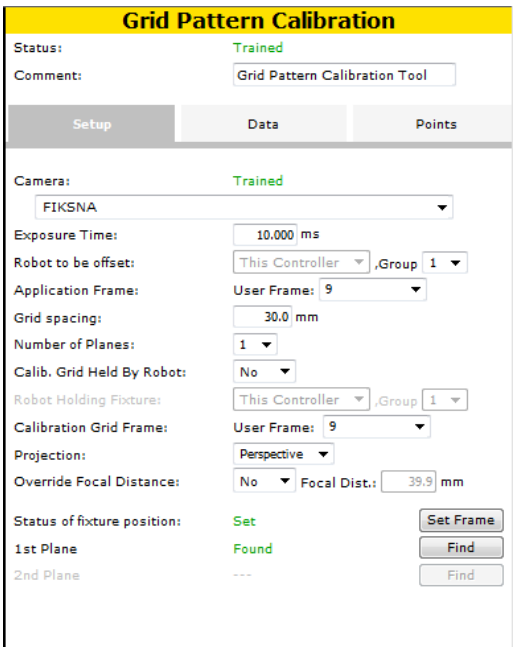
Unutar izbornika *Camera Calibration Tool* kreira se novi kalibracijski postupak *Grid Pattern Calibration* vrste. Zatim je potrebno postaviti nekoliko okvira. *Application Frame* i *Calibration Grid Frame*. Nakon što su sve stavke odabrane imamo kalibraciju koju kasnije možemo koristiti kod vizijskog procesa.

U izborniku je potrebno postaviti korisničko sučelje (*user frame*) definirano na robotu, zatim razmak objekata na mreži (*grid distance*) i fokalnu udaljenost (*focal distance*).



Slika 8 Grid Pattern Calibration

Za *User Frame* ćemo odabrati 9, a *grid distance* 30 mm. Kalibracija daje koordinate točaka na ekranu. Nakon toga spremimo kalibraciju pod imenom MIA kako bi je mogli koristiti prilikom pokretanja programa.

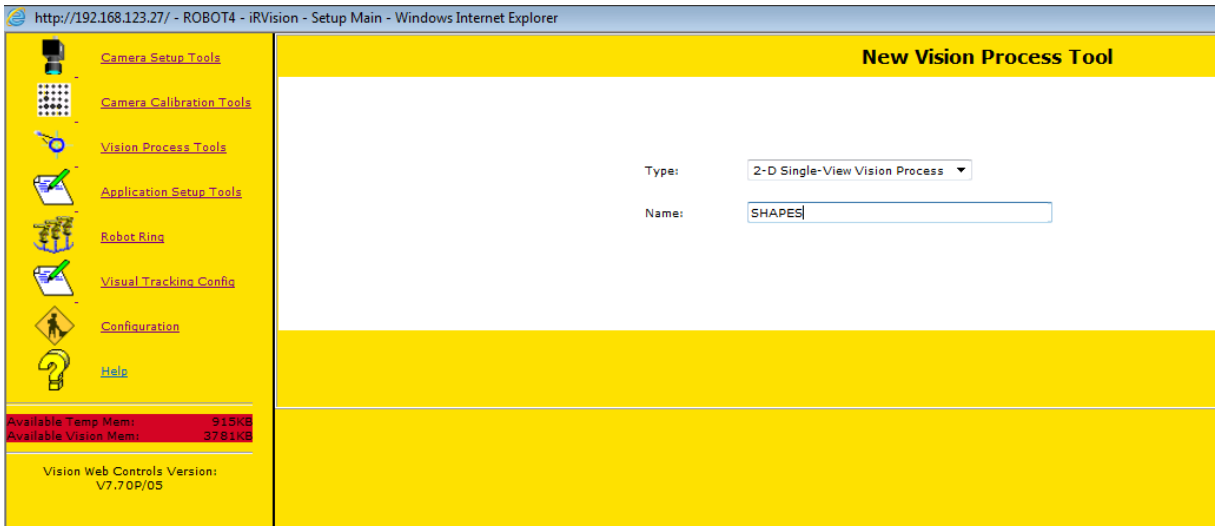


Slika 9 Postavke kalibracije

2.3. Program za prepoznavanje objekata

Kreiranjem novog 2D procesa vizije započinje postupak kojim ćemo podesiti postavke za detekciju i prepoznavanje objekata. Prva stavka u definiranju vizijskog procesa je kreiranje novih procesa traženja geometrijskih značajki – „Geometric pattern match tool“. Obrazac vizijski sustav uči tako da se u vidnom polju naznači koje područje se pretražuje te se uz pomoć algoritama za prepoznavanje dobivaju neke značajke predmeta. Tada se uz pomoć dodatnog alata pod nazivom „Training mask“ određuje koje se od pronađenih značajki neće koristiti pri pronalaženju predmeta. U početku postoji samo jedan *GPM Locator Tool* kojeg možemo podesiti tako da detektira određeni predmet. Potrebna su nam tri za tri različita oblika koja ćemo koristiti.

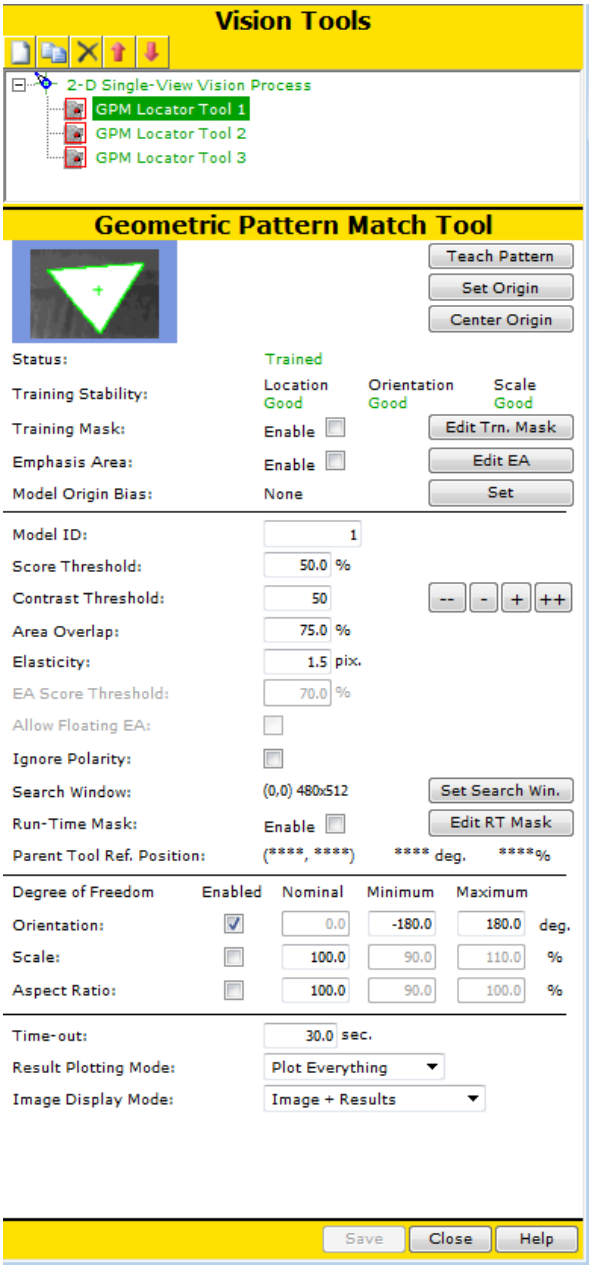
Postupak pokrećemo tako da kreiramo novi *Vision Process Tools* tipa *2-D Single-View Process* pod imenom SHAPES.



Slika 10 Kreiranje vizijskog procesa

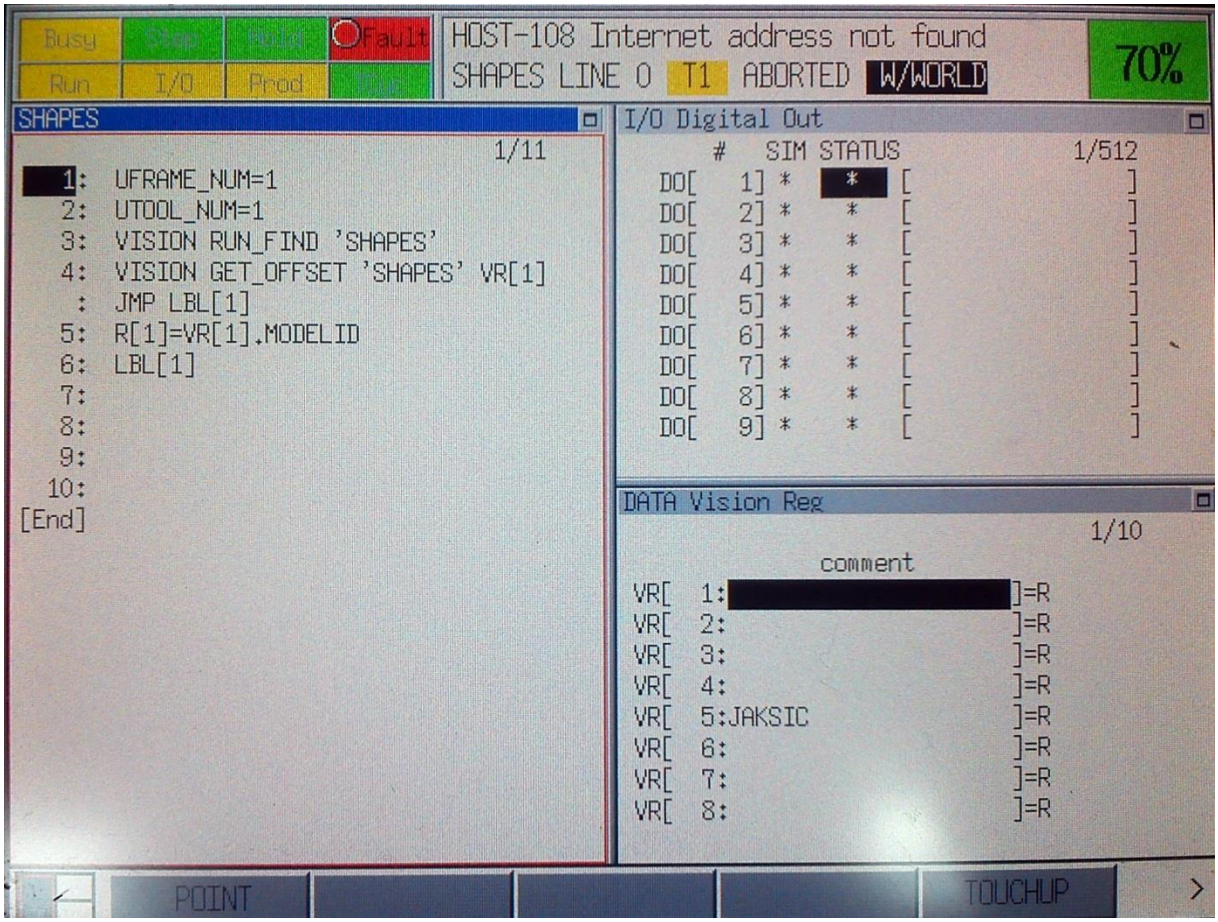
Nakon što je oblik postavljen pred kameru i slika prikazana na ekranu trebamo okinuti sliku kako bi na njoj odabrali stavku koju želimo kasnije koristiti za prepoznavanje. To učinimo klikom na *Snap* gumb i *Teach Pattern*. *Teach pattern* pamti oblik postavljen pred kameru, ali ga je najčešće potrebno doraditi kako bi šansa za prepoznavanje bila što veća. To znači da treba postaviti ishodište (*origin*) i izbrisati (*mask*) dijelove slike koji nam ne trebaju za prepoznavanje. Unutar ovog izbornika najvažnije je upisati ModelID. ModelID predstavlja identifikaciju na temelju koje će program kasnije raspoznati o kojem je obliku riječ.

Score Treshold definira sličnost između naučenog objekta onog slikanog za vrijeme procesa. Što je njegova vrijednost manja to je veća šansa da će vizijski proces prepoznati objekt pred kamerom. Ovdje odabiremo 50%. Nadalje potrebno je za svaki model postaviti njegov Model.ID. To radimo tako da u kućice upišemo vrijednosti od 1 do 3 za svaki model redom. Tako trokutu pripada vrijednost 1. Od drugih postavki ne treba ništa posebno mijenjati osim orijentacije za koju trebaju biti definirane vrijednosti od -180º do +180º.



Slika 11 Učenje vizijskog sustava

Program „SHAPES“ napisan na privjesku za učenje koristi nekoliko funkcija. Nakon definiranja korisničkog koordinatnog sustava i koordinatnog sustava alata potrebno je pokrenuti vizijski proces pozivom funkcije VISION RUN_FIND „SHAPES“. Funkcija VISION GET_OFFSET uzima i sprema vrijednost ofseta u poseban vizijski registar. Ako ništa nije pronađeno nakon pokretanja vizijskog procesa izlazi se iz programa koristeći JMP - LBL funkcije. Ova funkcija je potrebna kako bi vrijednost MODEL.ID mogli spremiti u određeni numerički registar koji ćemo kasnije pozvati u Karel kodu.



Slika 12 Program za pokretanje vizijskog sustava

2.4. Ponašanje robota

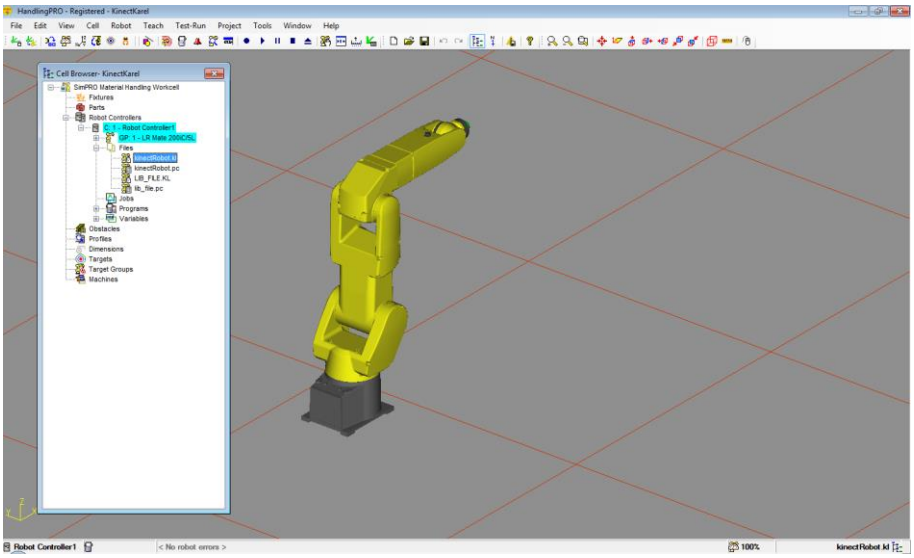
Ponašanje robota definirano je programskim kodom napisanim u programskom jeziku Karel koji je specifičan za Fanuc robote. Program obavlja dvije bitne zadaće, vrši komunikaciju između robota i računala te na temelju dobivenih točaka dovodi robota u željeni položaj. Komunikacija se odvija pomoću TCP/IP mrežnih komunikacijskih protokola, a svakih nekoliko desetaka milisekundi robot šalje svoj položaj računalu koje na temelju

pozicije robota i prepreka u radnom prostoru određuje sljedeću akciju koju zatim šalje nazad robotu. Računalo šalje robotu dvije glavne naredbe. Prva, koja započinje slovom „A“ nakon čega slijedi traženo stanje svih osi robota i brzina gibanja i druga koja započinje slovom „S“, a služi za zaustavljanje robota. Robot također šalje računalu naredbu koja započinje slovom „A“ nakon koje slijedi stanje za svaku os robota.

```
public void UpdatePosition()
        {
            string message = "A";

            foreach (AxisModel axis in _robotModel.Axes)
            {
                _robotModel.RobotState = RobotStates.Driving;
                message += axis.DesiredValue.ToString().Replace(',', '.') + ";";
            }

            message += Math.Round(_robotModel.Axes[0].Speed,
3).ToString().Replace(',', '.') + ";";

            SendMessage(message + "\n");
           }
```

2.4.1. ROBOGUIDE-HandlingPRO

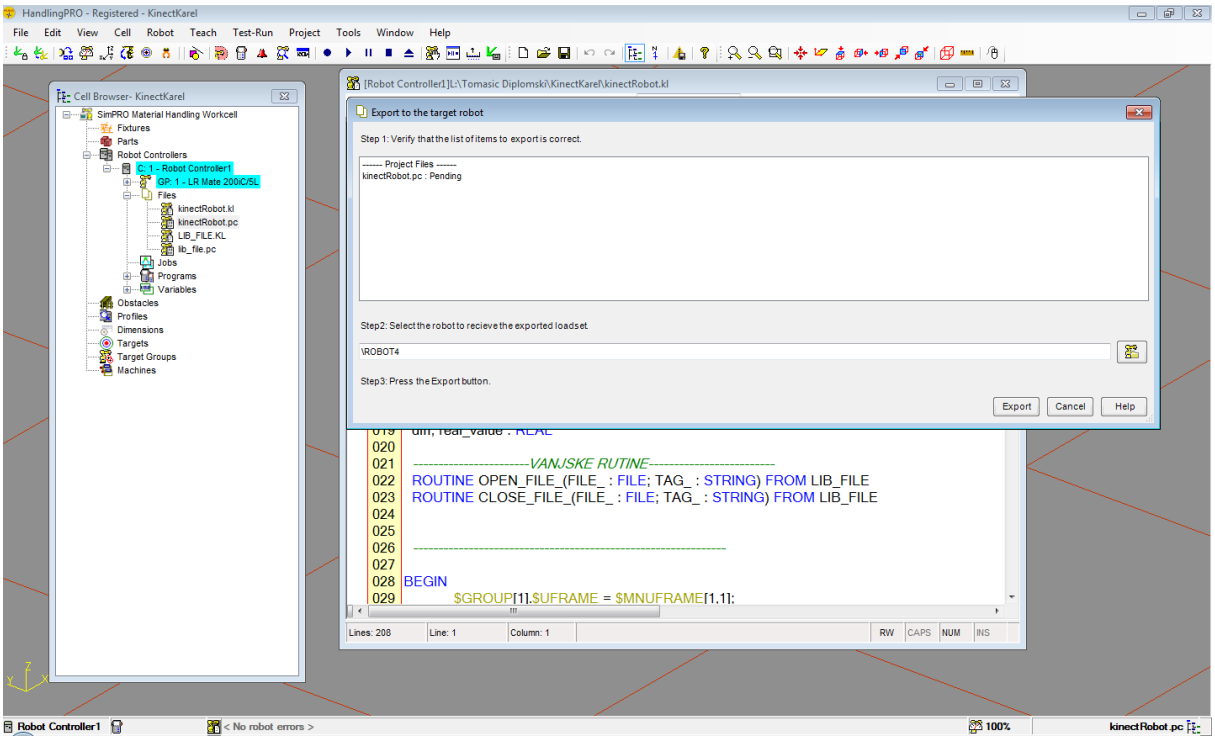
Pisanje Karel koda vrši se u programu ROBOGUIDE – HandlingPRO. HandlingPRO je član Fanuc-ove ROBOGUIDE familije proizvoda za offline simulaciju robota na virtualnom robotskom kontroleru.



Slika 13 ROBOGUIDE-HandlingPRO program

HandlingPRO omogućava korisniku simulaciju robotskog procesa u 3-D prostoru proučavanje istog. U ovom radu HandlingPRO je korišten samo kako bi se napisao kod koji će kasnije biti prebačen na robota.

Nakon što se kod napiše potrebno je kliknuti ikonu *Build* koja generira .pc file istog imena kao i Karel file. Taj file desnim klikom pa *Export-->To Robot* prebacujemo na privjesak za učenje.



Slika 14 Pisanje Karel koda

U Karel kodu se paralelno vrti i program koji poziva kod napisan na privjesku za učenje za pokretanje vizijskog sustava. Ponašanje robota ovisno o stanju vizijskog sustava također je definirano u Karel programskom jeziku.

Jednostavan program svako stoti korak postavlja registar na vrijednost 0, poziva program „SHAPES“ sa privjeska za učenje te uzima vrijednost numeričkog registra 1 koji sadrži podatke o tome koji je objekt detektiran. Objekti su definirani preko MODEL.ID-ja te je za prvi objekt (trokut) odabrana vrijednost 0, za drugi objekt (pravokutnik) vrijednost 1, a MODEL.ID trećeg objekta (termoregulator) je 3.

```
counter = counter + 1

                IF (counter = 100) THEN
                    counter = 0
                    start::
                    SET_INT_REG(1, 0, STATUS)
                    CALL_PROG ('SHAPES',prog_index)

                    GET_REG (1,real_flag,int_value,real_value,STATUS)
```

Funkcija SET_INT_REG postavlja cjelobrojnu vrijednost specifičnog registra. U ovom slučaju registar 1 je potrebno svaki put postaviti na nulu jer će u protivnom ostati zapamćeno prethodno stanje.

SET_INT_REG(register_no, int_value, status)

Ugrađena funkcija GET_REG ima pet argumenata od kojih ćemo koristiti samo *reg_no* i *int_value*. *register_no* označava registar iz kojeg ćemo dohvatiti podatke, a *int_value* daje informaciju o realnoj ili cjelobrojnoj vrijednosti numeričkog registra.

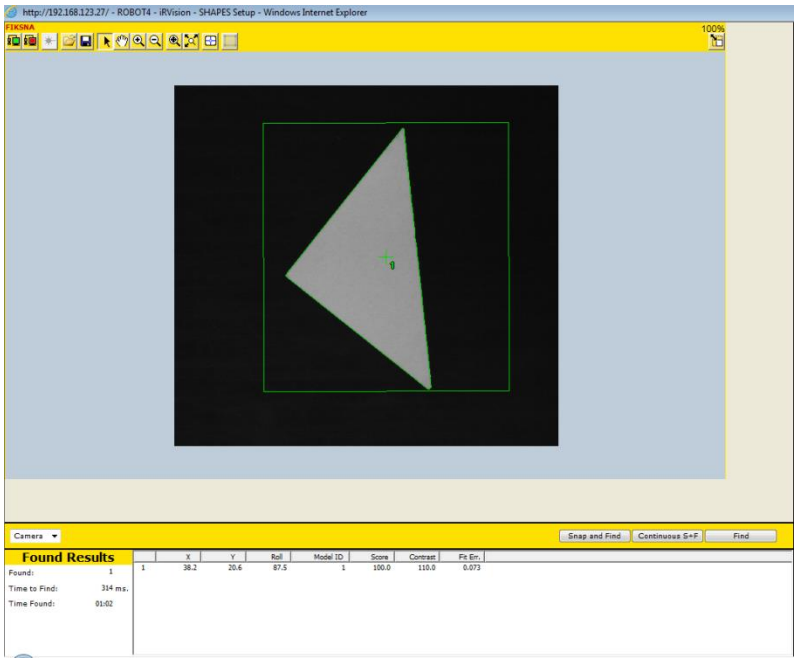
GET_REG(register_no, real_flag, int_value, real_value, status)

Sa tri IF petlje određeni su modeli ponašanja u ovisnosti u stanju registra.

```
IF (int_value)=1 THEN
                        DELAY 1000
                        GOTO start
                    ENDIF

                    IF (int_value)=2  THEN
                        MOVETO P4 NOWAIT
                    ENDIF

                    IF (int_value)=3 THEN
                        CALL_PROG('VPMIA',prog_index)
                    ENDIF

                    IF (int_value)<>0 THEN
                        P3 = CURPOS(0,0)
                        GOTO start
                    ENDIF
```

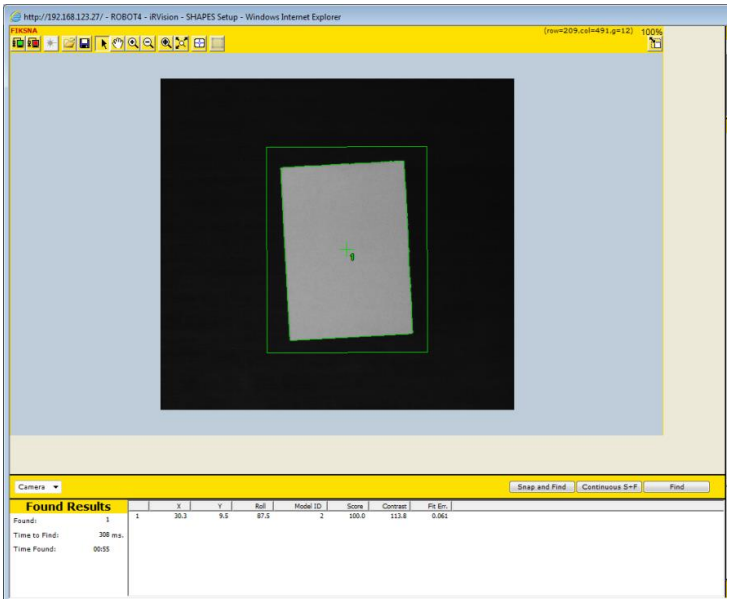
2.4.2. Načini ponašanja

Prvi model ponašanja zaustavlja robota kada se pred fiksnu kameru postavi trokutni oblik. Nakon micanja trokutnog oblika robot nastavlja gibanje koje mu je prije definirano Karel kodom. To je riješeno na pomoću naredbe DELAY koja zaustavlja izvršavanje programa određeno vrijeme izraženo u milisekundama. Nakon 1000 milisekundi program se vraća na početak i ponovno provjerava stanje registra.



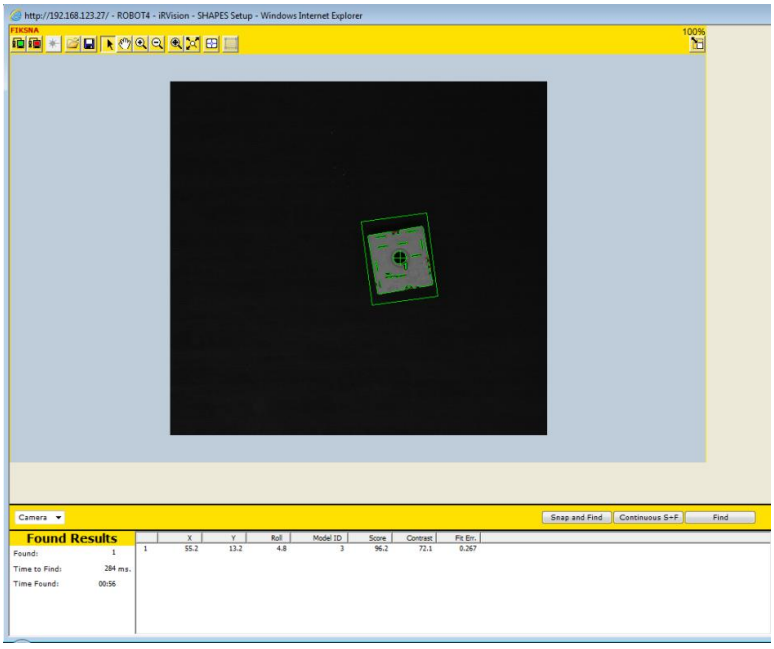
Slika 15 Prvi objekt za prepoznavanje-trokut

Drugi model ponašanja šalje robota u prije definiranu točku P4 nakon koje je potrebno dodati zastavicu. Dodavanjem zastavice NOWAIT naredbi za izvršavanje gibanja sprječava se blokiranje programa i izvršavanje naredbi se nastavlja neovisno o gibanju robota. No, ako se nakon jedne naredbe gibanja sa NOWAIT krene izvršavati sljedeća naredba gibanja, ona opet blokira program sve dok se ne izvrši prošlo gibanje. Drugi model ponašanja se aktivira ako je pred kameru postavljen pravokutni oblik.



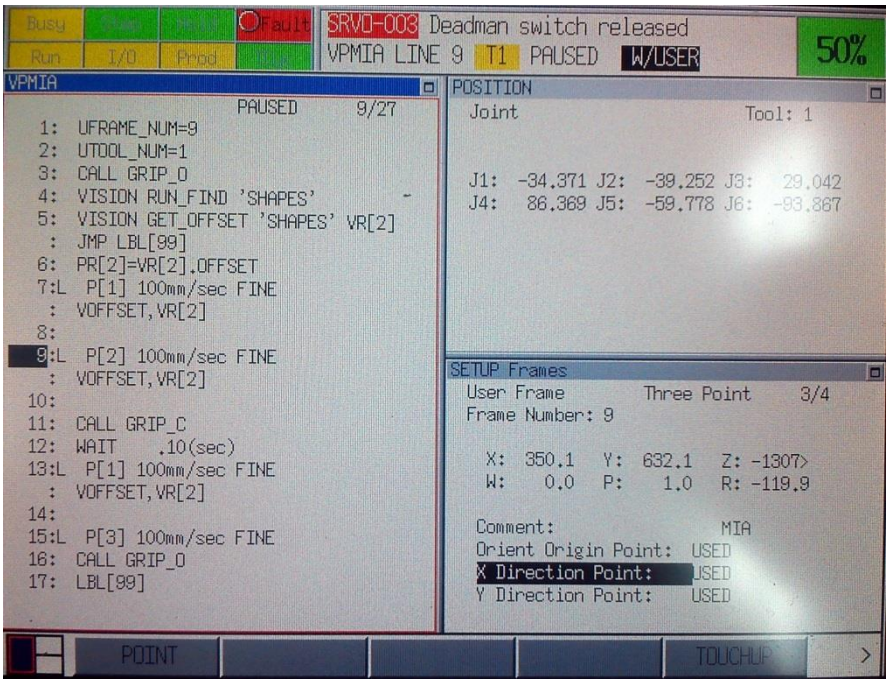
Slika 16 Drugi objekt za prepoznavanje-pravokutnik

Treći model ponašanja uključuje pozivanje programa sa privjeska za učenje pod nazivom „VPMIA“. Taj potprogram služi za obavljanje jednostavne *Pick and Place* radnje, a aktivira se ako kamera detektira termoregulator. Robot tada prekida svoje gibanje, uzima termoregulator, odlaže ga u za to predviđenu kutiju i nastavlja svoje gibanje koje je obavljao prije prekida.



Slika 17 Treći objekt za prepoznavanje-termoregulator

Kod napisan na privjesku za učenje:



Slika 18 Kod za *Pick and Place* radnju napisan na privjesku za učenje

2.5. Problemi pri izradi programa

Glavni problem pri izradi programa je bio taj što robot kada mu se zadaju točke A i B započne svoje kretanje i program se blokira sve dok se to gibanje ne izvrši. Kako bi robot mogao prekinuti svoje gibanje i promijeniti ponašanje u ovisnosti o stanju vizijskog sustava te kasnije nastaviti isto osmišljen je sustav koji svako gibanje od točke A do točke B rastavlja na nekoliko manjih segmenata od nekoliko centimetara.

Kada se takav program piše na privjesku za učenje može se isprogramirati gibanje na način da se kreira pozicijski registar koji se stalno pomiče za jedan segment koristeći funkcije OFFSET i TOOL OFFSET. OFFSET je naredba kojim možemo jednu točku, više točaka ili cjelokupan program pomaknuti bez promjene programa, a TOOL OFFSET je naredba kojom možemo odrediti odmak pozicije robota od zadane točke i to s obzirom na koordinatni sustav alata.

Ako program pišemo u Karel-u rastavljanje na segmente se vrši tako da se robot stalno pomiče za neki diferencijal definiran u ovisnosti o brzini gibanja. Zbog načina na koji robot planira putanju gibanja, ako je put gibanja prekratak on će na tom putu koristiti brzinu manju od nominalne. Što je veći broj segmenata, robot može brže prilagoditi putanju ovisno o stanju radne okoline, ali se time smanjuje brzina gibanja. Zbog toga je za neku željenu brzinu

potrebno odrediti optimalan broj segmenata, kako bi se ostvarila što veća brzina uz zadovoljavajuću brzinu reakcije na promjene putanje.

Vizijski sustav se iz istog razloga provjerava svako stoti korak koristeći poseban *counter*. Vrijednost sto je također određena eksperimentalno. Za manje vrijednost gibanje robota nije kontinuirano već isprekidano pa postaje očito da je putanja podijeljena na više dijelova.

3. IMPLEMENTACIJA PROGRAMA NA FANUC M – 10*i*A

U dinamičnoj okolini u kojoj robot i čovjek istovremeno surađuju na obavljanju zadatka, potrebno je poznavati položaj čovjeka u prostoru, kako bi se izbjegle eventualne ozljede, te ubrzalo izvođenje zadatka.

Za dobivanje položaja čovjeka i ostalih prepreka u radnom prostoru upotrijebiti će se Microsoftov uređaj Kinect. Kako sustav mora biti u mogućnosti izbjeći ne samo čovjeka, već i predmete koji mu se postave u radni prostor, neće se moći koristiti podaci o položaju zglobova čovjeka, već će se eventualni sudari sa preprekama izračunavati iz dubinske mape dobivene od Kinect-a.

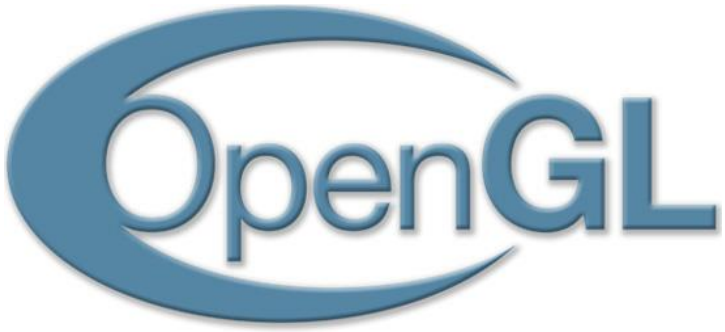
Kako bi se povećao pregled nad radnim prostorom i izbjegla situacija u kojoj se robot nalazi između prepreke i Kinect-a, koristiti će se dva Kinect-a. Time je u normalnim situacijama kada nema preklapanja sa robotom dodan stupanj redundancije koji dodatno osigurava detekciju prepreke.

U izradi rada korišten je Fanuc M3-*i*A robot delta strukture. Iako brz ovaj robot zauzima veliki volumen. Sustav zanemaruje prostor koji robot zauzima, iz njega ne dobiva korisne informacije. Zbog složenosti strukture i njene izmjene u različitim položajima robota zanemaren je znatno veći volumen prostora nego što bi to bilo potrebno sa klasičnom strukturom robota.

Prednost klasične strukture nad delta strukturom je upravo u tome što takav robot zauzima puno manji volumen i zbog činjenice da je takva struktura puno češće korištena program je implementiran i na klasičnu strukturu. Ono što je bitna razlika u odnosu na delta strukturu je u tome što osim položaja TCP (*Tool center point*) i brzine, robot šalje i vrijednosti svojih unutarnjih koordinata te se na temelju tih informacija iscrtava struktura u OpenGL kontroli.

3.1. OpenGL

OpenGL je standardno sučelje namjenskih programa (API) za definiranje 2-D i 3-D grafičkih slika. Prije OpenGL-a svaka tvrtka koja je razvila grafički program obično je morala prepisati grafički dio za svaku platformu operacijskog sustava. S OpenGL-om program može napraviti isti efekt u bilo kojem operacijskom sustavu koristeći bilo koji grafički adapter koji podržava OpenGL.



Slika 19 Logotip OpenGL-a

Podržava više programskih jezika i rad na različitim platformama. Za C# postoji više wrappera (sučelja prema C++ funkcijama OpenGL-a), u ovom radu koristiti će se wrapper OpenTK.

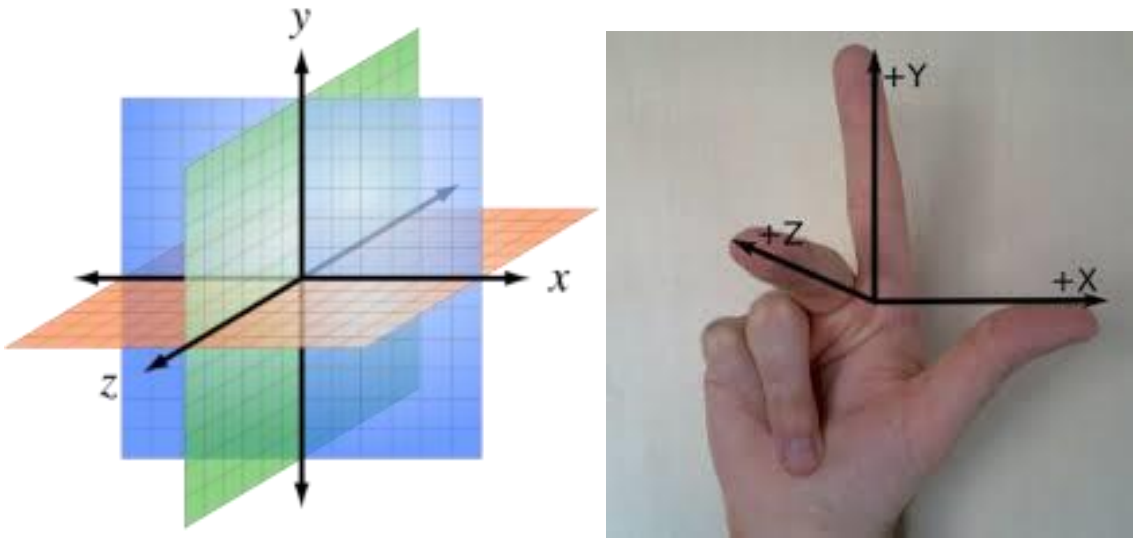
OpenGL je razvijen od strane Silicon Graphics Inc. (SGI) 1991. i naširoko se koristi u CAD sustavima, virtualnoj stvarnosti, vizualizaciji, simulacijama i video igrama. Prvenstveno je razvijen kako bi se premostio jaz između hardvera i softvera te da se olakša i pojednostavni kod za pisanje 3-D aplikacija. Pomoću OpenGL-a složeni trodimenzionalni oblici se lako prikazuju koristeći jednostavne geometrijske oblike.

Bez hardverskih ubrzivača mnoge mogućnosti OpenGL-a bi radile jako sporo pa OpenGL koristi specijalizirane mogućnosti novih 3-D video kartica za vrlo brzo renderiranje 3-D grafike.

3.1.1. Koordinatni sustav i transformacije

Grafički sustavi aplikacija definirani su koordinatnim sustavom prema pravilu desne ruke. Kod koordinatnog sustava desne ruke, gledano iz pozitivnog kraja osi, pozitivna rotacija oko osi se izvodi suprotno od kazaljke na satu, Z koordinata određuje dubinu objekta, Y visinu, a X širinu.

Glavne rutine za transformaciju modela su: `GL.Translate()`, `GL.Rotate()` i `GL.Scale()`. Te naredbe transformiraju objekt (ili koordinatni sustav) tako da ga pomiču,rotiraju, istežu, skupljaju ili ga reflektiraju. Sve tri naredbe u biti automatski izračunavaju odgovarajuću matricu translacije, rotacije i skaliranja iz parametara poslanih u pozivu funkcije.



Slika 20 Koordinatni sustav OpenGL-a

3.1.2. Osnovni oblici

OpenGL se često naziva API niske razine zbog minimalne podrške za geometrijske oblike višeg reda, strukture podataka kao što su grafovi scene ili podrške za učitavanje 2D slikovnih datoteka ili datoteka 3D modela. Umjesto toga OpenGL se fokusira na učinkovito renderiranje osnovnih geometrijskih oblika niske razine sa različitim osnovnim, ali fleksibilnim postavkama za renderiranje.

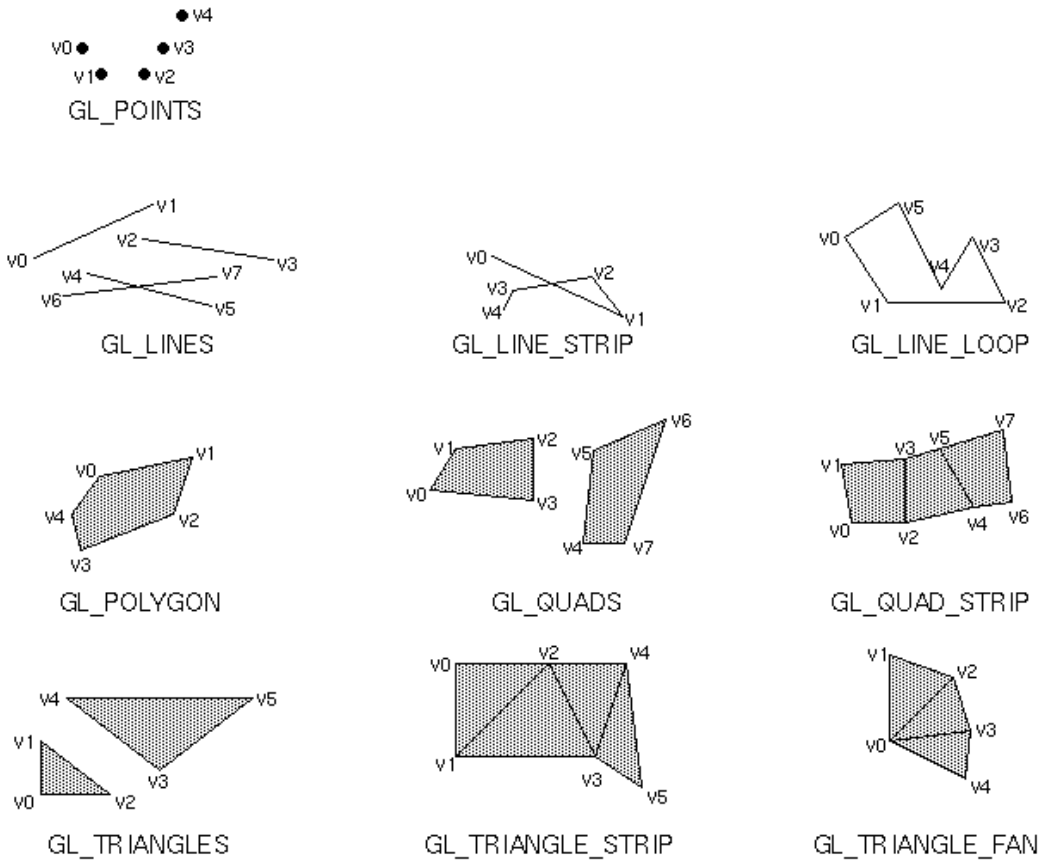
OpenGL aplikacije renderiraju osnovne geometrijske oblike tako da odrede tip osnovnog oblika i redoslijed vrhova s pripadajućim podacima. Tip osnovnog geometrijskog oblika određuje kako OpenGL tumači i renderira niz vrhova. OpenGL pruža deset različitih osnovnih tipova za crtanje točke, linije i poligona.

OpenGL aplikacije renderiraju osnovne geometrijske oblike po danim vrhovima sa parom funkcija `GL.Begin()` i `GL.End()`. Aplikacija određuje tip geometrijskog oblika tako da ga propušta kao parametar u funkciju `GL.Begin()`.

Funkcije `GL.Begin()` i `GL.End()` su naredbe koje specificiraju početak i kraj prikaza vrhova. Parametar „mode“ određuje tip osnovnog geometrijskog oblika koji će se crtati. Između naredbi `GL.Begin()` i `GL.End()` šalju se vrhovi i stanja vrhova kao što su trenutna primarna boja, trenutna normala, svojstva materijala za rasvjetu, te trenutne koordinate teksture za mapiranje tekstura.

3.1.3. Crtanje osnovnih oblika

Kako bi nacrtali složene 3-D oblike kao što je radni prostor,robot i putanja i prikazali ih potrebno je koristiti određene OpenGL naredbe za crtanje. OpenGL se naziva API niske razine upravo zbog minimalne podrške za učitavanje 2-D ili 3-D modela. Umjesto toga koncentrira se na učinkovito renderiranje osnovnih geometrijskih oblika na način da odredi tip osnovnog oblika i redoslijed vrhova s pripadajućim podacima.



Slika 21 OpenGL osnovni tipovi geometrijskih oblika

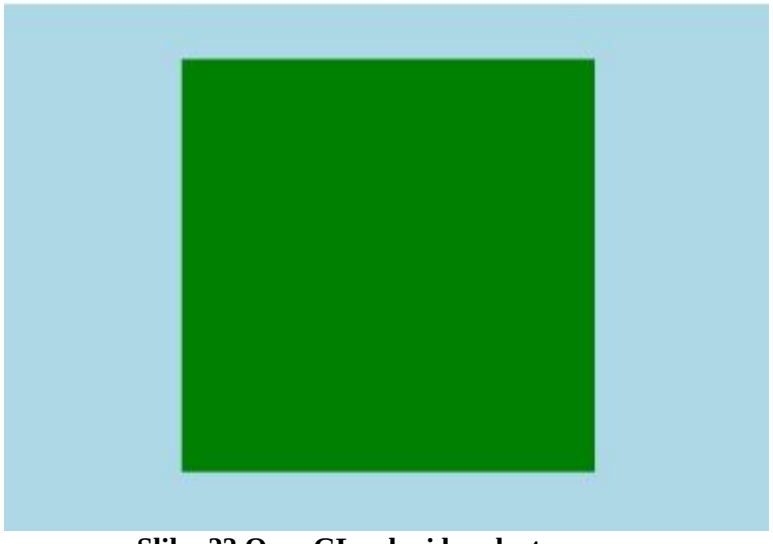
OpenGL pruža deset različitih osnovnih tipova za crtanje točke,linije i poligona. To su :

- GL_POINTS
- GL_LINES
- GL_LINE_STRIP
- GL_LINE_LOOP
- GL_TRIANGLES
- GL_TRIANGLE_STRIP
- GL_TRIANGLE_FAN
- GL_QUADS
- GL_QUAD_STRIP
- GL_POLYGON

Za crtanje radnog prostora, robota i putanje koristit će se funkcije *DrawRectangle* i *DrawPolygonRectangle* od čega prva koristi GL_LINES za crtanje kvadra određenih dimenzija, a druga koristi GL_QUADS za crtanje punih kocki određene dimenzije i boje.

[3] Na primjer da bi se nacrtao zeleni kvadrat stranica jedinične vrijednosti koristi se sljedeći kod:

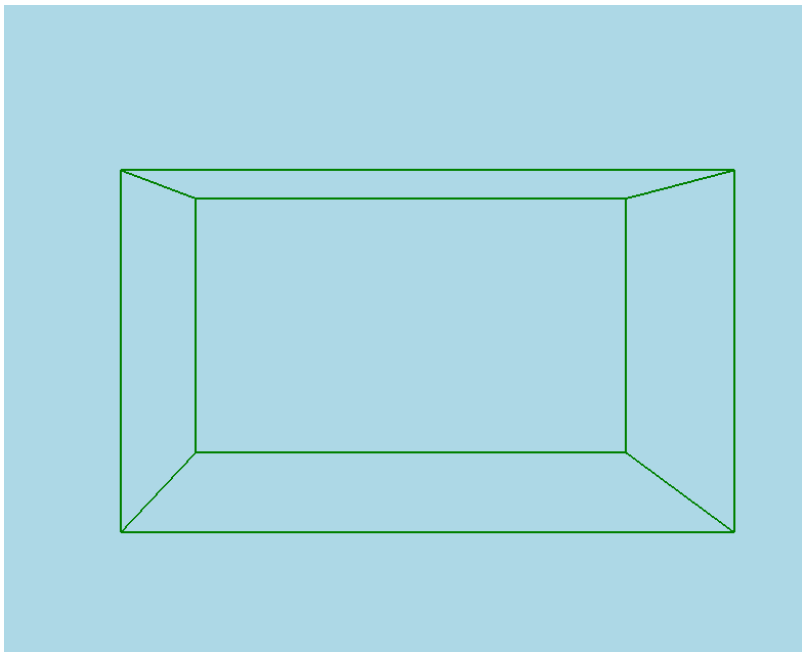
```
GL.Begin(BeginMode.Quads);
GL.Color3(System.Drawing.Color.Green);
GL.Normal3(-1.0f, 0.0f, 0.0f);
GL.Vertex3(0.0f, 0.0f, 0.0f);
GL.Vertex3(0.0f, 0.0f, 1.0f);
GL.Vertex3(1.0f, 0.0f, 1.0f);
GL.Vertex3(1.0f, 0.0f, 0.0f);
GL.End();
```



Slika 22 OpenGL zeleni kvadrat

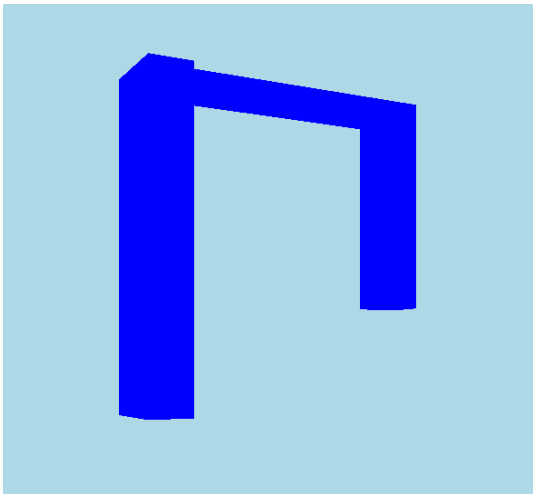
3.2. Radni prostor

Prostor u kojem može doći do interakcije robota i čovjeka ili robota i nekog predmeta koji se tamo nalazi, nazvan je radni prostor. Radni prostor je kvadar proizvoljnih dimenzija širine, visine i dubine u kojem robot odrađuje koristan rad, poput premještanja predmeta, zavarivanja, bojanja, ili samo prolazi kroz taj prostor kako bi došla do točke rada koja je izvan prostora interakcije. U njemu se nalaze dinamični ili statični objekti koji svojim položajem i/ili gibanjem mogu zasmetati robotu u izvršenju njegovih zadataka.



Slika 23 OpenGL radni prostor robota

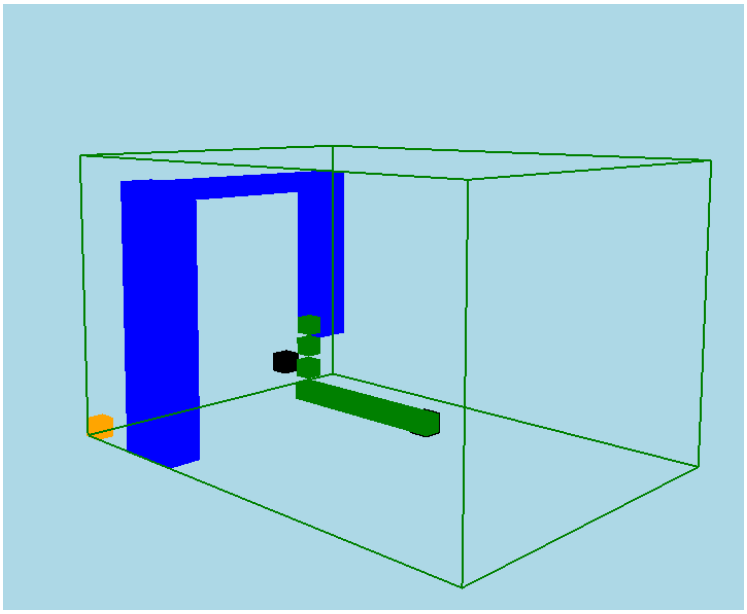
Radni prostor podijeljen je na kockice jednake duljine, visine i širine definirane od strane korisnika. U programu je također definirana i duljina, visina i širina samog radnog prostora. Podjela na kockice odnosno voksele služi kako bi se dobila realna slika treće dimenzije odnosno dubine kako za strukturu robota tako i za položaj prepreke ili objekta koji može zasmetati gibanju robota. Radni prostor se crta pomoću OpenGL-a gdje se plavom bojom prikazuju vokseli koji sadrže robota, crvenom oni koji sadrže prepreku, a zelenom zadana putanja po kojoj se robot treba gibati.



Slika 24 OpenGL robot

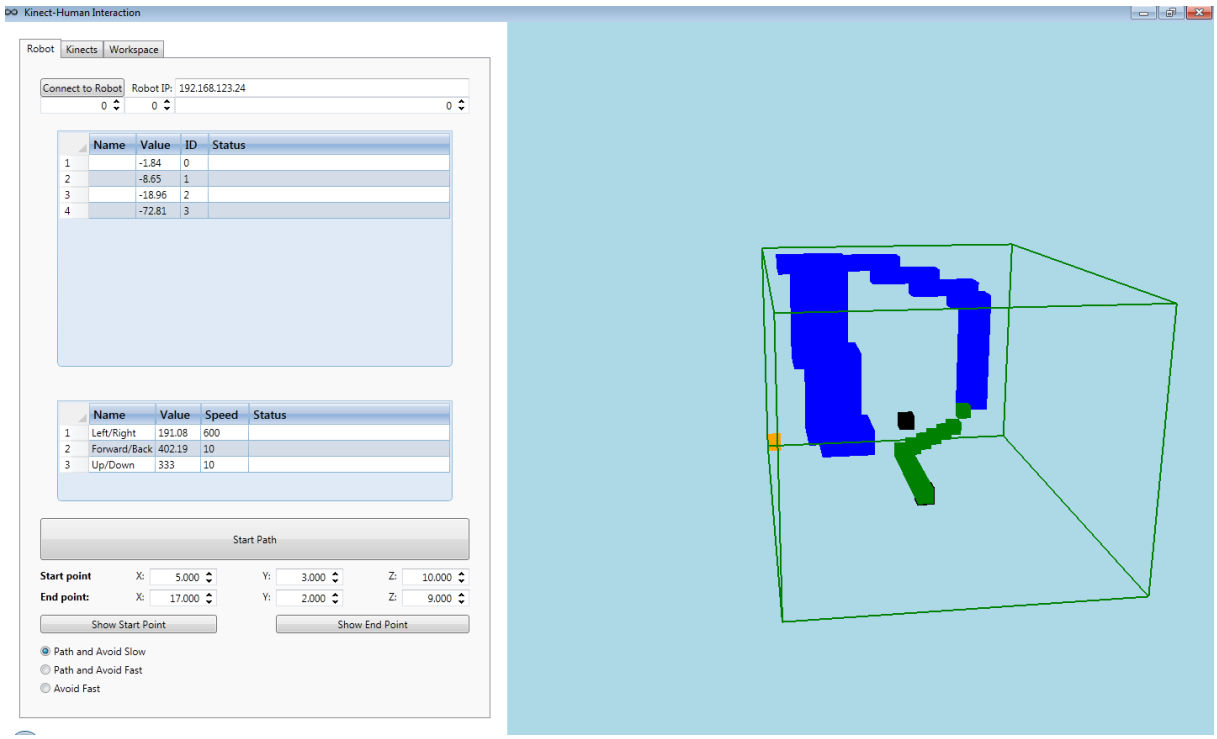
3.3. Robot u radnom prostoru

Izgled robota u radnom prostoru prikazan je pomoću OpenGL sučelja za programiranje pomoću kojeg se crta izgled robota za svaku poziciju u svakom trenutku. Potrebno je nacrtati robota koristeći podatke o trenutnoj poziciji prihvatnice i kutovima zakreta u zglobovima. Korištena su četiri kuta budući da se to pokazalo sasvim dovoljnim za vjeran prikaz robota u radnom prostoru. Osim tih pet podataka potrebno je i odrediti ofset robota koji se namješta u odnosu na trenutnu poziciju prihvatnice i kutova.



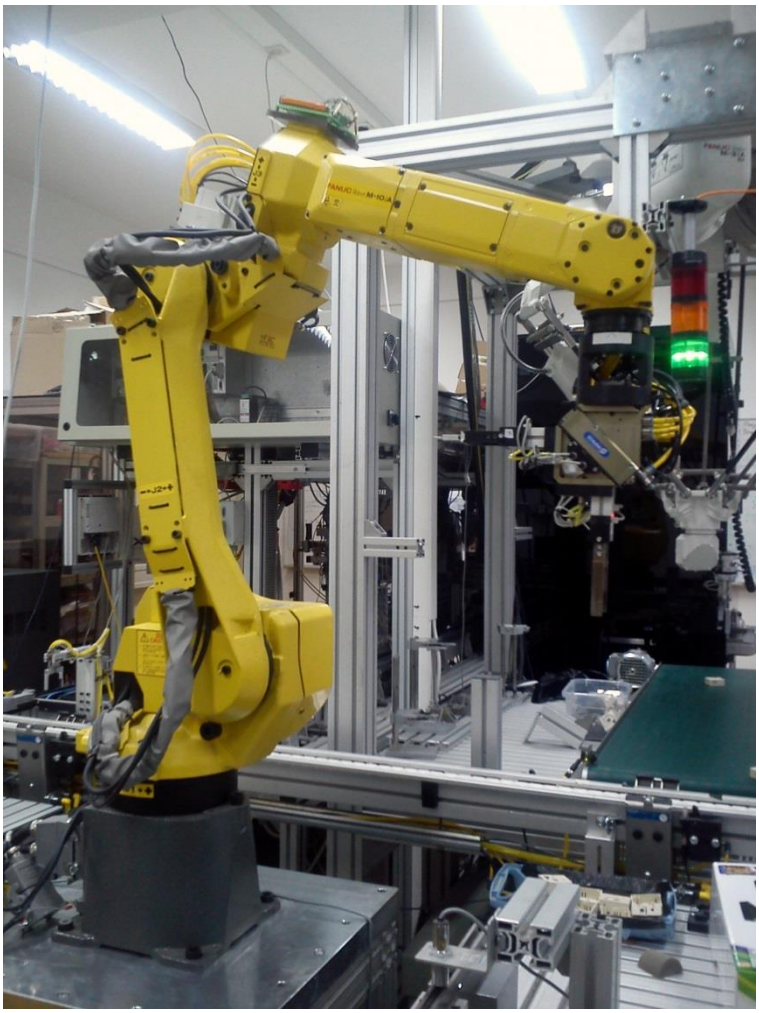
Slika 25 OpenGL robot u radnom prostoru

Iscrtavanje robota vrši se pomoću dvije funkcije. Funkcije za dobivanje x, y i z koordinata (*FindSquaresWithRobotNewStructure*) i funkcije koja iscrtava robota s obzirom na te koordinate (*PlaceRobotAround*).



Slika 26 OpenGL robot u prostoru za određenu konfiguraciju

Izgled robota u prostoru:



Slika 27 Stvarni izgled robota u prostoru

Prva funkcija koristi vrijednosti koordinata prihvatnice i kutove zakreta zglobova. Tri podatka o poziciji prihvatnice označavaju vrh robota i početak putanje. Kutovi zakreta potrebni za iscrtavanje se određuju potrebnim transformacijama (rotacija za 90° ili 180°) i prebacivanjem u radijane umjesto u stupnjeve. Robot klasične strukture se crta u tri dijela. Prvi najširi je duljine 650 i radijusa 2, dok su srednji dio duljine 550 i zadnji (koji sadrži prihvatnicu) radijusa 1. Nakon definicije lokalnih varijabli potrebno je izračunati sinuse i kosinuse kutova. Dobiveni podaci prije svega služe za izračunavanje ofseta na način da se od koordinate prihvatnice oduzme ukupna duljina robota za x, y i z vrijednost. Kutovi također služe kako bi se vršilo iscrtavanje po svakoj duljini zasebno.

```
int a = (int)(_robotViewModel.Axes[0].Value / workSpace.elementSize + 20f);
int c = (int)(_robotViewModel.Axes[1].Value / workSpace.elementSize + 5);
int b = (int)(_robotViewModel.Axes[2].Value / workSpace.elementSize - 5f);

(…)

robotAngle0 = (_robotViewModel.Angles[0].Value+90) / 180.0 * 3.14;
robotAngle1 = (90 - _robotViewModel.Angles[1].Value) / 180.0 * 3.14;
robotAngle2 = (_robotViewModel.Angles[2].Value) / 180.0 * 3.14;
robotAngle3 = ((-_robotViewModel.Angles[3].Value - 180)-RobotAngle2) / 180.0 *
3.14;


int L1 = 650;
int L2 = 550;
int L3 = 350;
```

Druga funkcija se poziva unutar svake od tri *for* petlje koja počinje od nule i završava sa vrijednosti L1, L2 odnosno L3. Unutar tih petlji se računaju x, y i z vrijednosti kvadrata koji sadrže robota koristeći izračunate podatke o sinusima i kosinusima kutova. Ta funkcija svaku dobivenu točku označava kao *true* za sadržavanje robota. Ukoliko točka sadrži robota ona se pomoću OpenGL potrebnim transformacijama koordinatnog sustava iscrtava plavom bojom, dok se putanja koja počinje od vrha prihvatnice iscrtava zelenom bojom.

```
public void PlaceRobotAround(int xSquare, int ySquare, int zSquare, int radius)
        {
            for (int i = xSquare - radius; i < xSquare + radius; i++)
            {
                for (int j = zSquare - radius; j < zSquare + radius; j++)
                {
                    if (!(i < 0 || i > workSpace.width / workSpace.elementSize - 1
||
                            ySquare < 0 || ySquare > workSpace.height /
workSpace.elementSize - 1 ||
                            j < 0 || j > workSpace.depth / workSpace.elementSize -
1))
                    {
                        workSpace.elements[i, ySquare, j].hasRobot = true;
                    }
                }
             }

            }
```

3.4. Uključivanje robota u sustav

U ovom radu korišten je robot Fanuc M – 10*i*A klasične strukture. To je šestoosni robot težine 130 kg koji iako je malen, ali dosta moćan omogućuje i do 10 kg korisne nosivosti. Ovaj robot za razliku od M3 – *i*A robota zauzima manji volumen i relativno je jednostavnije strukture pa je izbjegnuta potreba za zanemarivanjem većeg radnog prostora nego što je to potrebno.

Slika 28 Fanuc M3-*i*A

3.4.1. Program na robotu

Program na robotu napisan je u programskom jeziku Karel koji je specifičan za Fanuc robote kao i za delta strukturu tako i za klasičnu strukturu. Komunikacija između računala i robota se također vrši pomoću TCP/IP komunikacijskih protokola, a robot osim što šalje informaciju o trenutnom stanju osi i brzini šalje i informaciju o stanju unutarnjih koordinata. Računalo vraća nazad robotu željene vrijednosti za osi, odnosno položaj alata.

Vrijednosti unutarnjih koordinata odnosno kutova q1,q2,q3 i q4 se spremaju u registar broj 1 (*reg_no=1*), te se za dohvaćanje trenutnog stanja kutova koristi funkcija CURJPOS. Funkcija CURJPOS vraća trenutno stanje svih zglobova s obzirom na TCP.

CURJPOS(axs_lim_mask, ovr_trv_mask <,group_no>)

```
reg_no=1
jpos=CURJPOS(0,0)
```

axs_lim_mask označava koje su osi izvan limita,a *ovr_trv_mask* označava amplitudu gibanja strojnog dijela izvan granica potrebnog da obavi traženu funkciju. Ovdje je korištena sintaksa gdje su *axs_lim_mask* i *ovr_trv_mask* jednake nuli.

Nakon toga korištenjem funkcije SET_JPOS_REG spremaju se vrijednosti dobivene funkcijom CURJPOS u prije definirani registar 1.

SET_JPOS_REG(register_no, jpos, status<, group_no>)

Slična funkcija GET_JPOS_REG dohvaća vrijednosti iz pozicijskog registra i pridružuje ih varijabli tipa JOINTPOS. Da bi dobili informacije o svakom od kutova potrebno je kod definicije varijabli odrediti jednu tipa ARRAY[6] OF REAL u koju će se spremiti sve informacije koje ćemo kasnije moći izvući. Prebacivanje u takav tip varijable vrši se funkcijom CNV_JPOS_REL.

CNV_JPOS_REL(jointpos, real_array, status)

```
SET_JPOS_REG(reg_no,jpos,STATUS)
cur_pos=GET_JPOS_REG(1,STATUS)
CNV_JPOS_REL(cur_pos,jp_org,STATUS)
```

Slanje vrijednosti kutova vrši se tako da se realni broj prebacuje u string koji se potom šalje računalu u obliku poruke. Svakoj varijabli dodijeljena je vrijednost kutova redom od 1 do 4. Funkcija koja to omogućava zove se CNV_REL_STR,

CNV_REAL_STR(source, length, num_digits, target)

gdje je *source* polje od šest vrijednosti,a *target* string varijabla.

```
CNV_REAL_STR(jp_org[1],2,2, q1)

CNV_REAL_STR(jp_org[2],2,2, q2)

CNV_REAL_STR(jp_org[3],2,2, q3)

CNV_REAL_STR(jp_org[5],2,2, q4)

WRITE file_var ('Q[0]=' +q1 + ';Q[1]=' + q2 + ';Q[2]=' + q3 + ';Q[3]=' +
q4+ ';', CR)
```

3.4.2. Program na računalu

Program na računalu napisan u C# programskom jeziku služi kako bi se podaci dobiveni iz Karel-a na adekvatan način primijenili za izračun položaja prihvatnice i kutova zakreta zglobova. Zbog toga je program u C# napisan prema MVVM (Model-View-ViewModel) uzorku programiranja čime se ostvaruje čist i pregledan kod.

Važno je spomenuti *AxesListViewModel* i *AngleListViewModel* koji se pozivaju unutar konstruktora *RobotViewModel* potprograma. Unutar istoga definirane su tri osi i četiri kuta kojima je pridružena vrijednost. Svaka od osi i svaki od kutova ima svoje ime (*Name*), identifikaciju (*ID*) i vrijednost (*Value*). Svaki od podataka se prepoznaje pomoću svog ID-ja koji ja za osi od nula do tri i za kutove od nula do četiri.

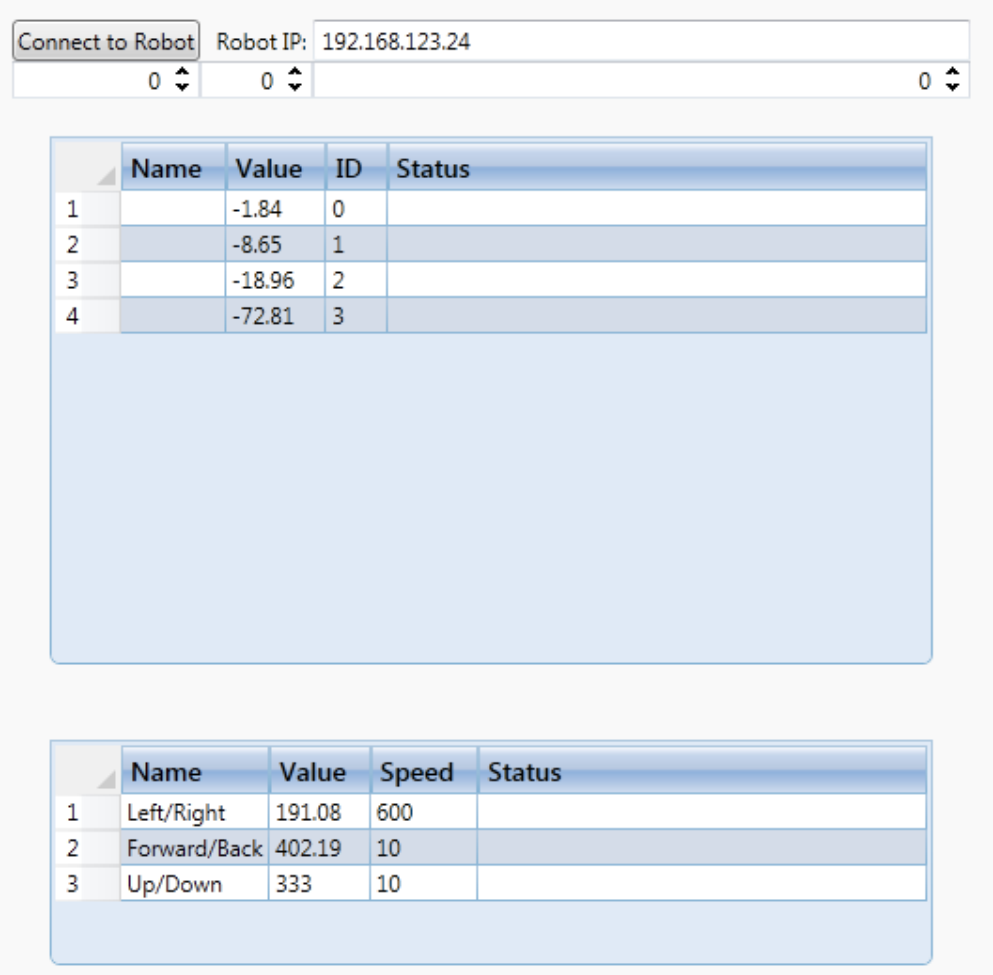
```
namespace Robot
{
    public class AngleListViewModel : Utilities.ViewModelBase
    {
        RobotViewModel _robotViewModel;

        public RobotViewModel MainpulatorViewModel
        {
            get { return _robotViewModel; }
        }

        public AngleListViewModel(RobotViewModel robotViewModel)
        {
            _robotViewModel = robotViewModel;
        }
    }
   }

   (…)

AngleModel kut1 = new AngleModel();
kut1.Value = 0;
kut1.ID = 0;
```

Te vrijednosti su upisane unutar poruke poslane računalu od strane robota putem Karel koda.

```
else if (statusMessage[0] == 'Q')
                                {
                                    int startIndex = statusMessage.IndexOf("[") + 1;
                                    int endIndex = statusMessage.IndexOf("]",
startIndex);
                                    int angleID =
int.Parse(statusMessage.Substring(startIndex, endIndex - startIndex));

                                    AngleModel angle =
_robotModel.Angles.FirstOrDefault(a => a.ID == angleID);
```

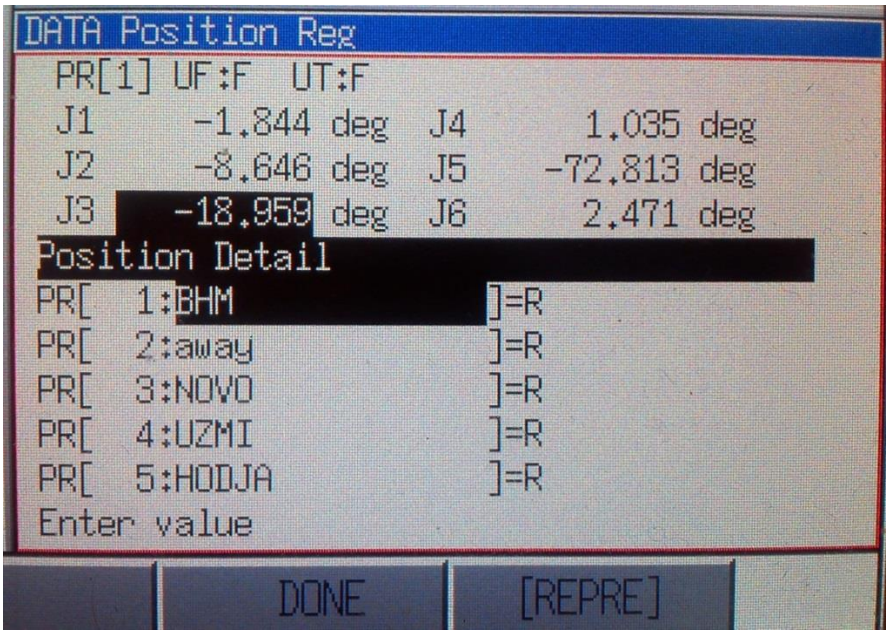
```
                                string value = statusMessage.Substring(endIndex +
2).Trim(new char[] { '\n', ' ' });//.Replace('.', ',');
                                double angleValue = double.Parse(value);

                                angle.Value = angleValue;

                               }
```

Dobiveni podaci se mogu prikazati unutar dvije tablice dizajnirane u .xaml dijelu koda. Na taj način možemo u svakom trenutku pratiti stanje osi i zglobova.



Slika 29 Dvije tablice za prikaz stanja kutova i prihvatnice

Kutovi na privjesku za učenje:



Slika 30 Kutovi na privjesku za učenje

3.5. Problemi pri izradi programa

Iako je relativno lako dohvatiti podatke o trenutnoj poziciji robota, problem nastaje kada želimo vjerno prikazati izgled i strukturu robota na računalu.

Naime, robota možemo nacrtati tako da odredimo duljine 3 segmenta i s obzirom na kutove zakreta nacrtamo robota koristeći OpenGL i prije spomenute funkcije. Ono što tada ne znamo je točna pozicija prihvatnice koja nam treba budući da računalo robotu šalje koordinate željene pozicije u odnosu na prihvatnicu. Jednostavnije rečeno, ako ćemo robota iscrtavati od postolja prema prihvatnici, vrh robota od kojeg kreće putanja i realna pozicija prihvatnice neće se podudarati pa gibanje robota neće biti kontinuirano ni točno.

Drugi način je crtati robota od prihvatnice prema postolju. Ovaj način ispostavio se loš zato što cijeli strukturu moramo crtati obrnuto i postolje nije učvršćeno već se pomiče kako se i robot pomiče.

Treći isproban način je pokazao najbolje rezultate. Ofset robota se određuje na temelju koordinata prihvatnice. Prvo se dobiju x, y i z koordinate prihvatnice, a zatim se izračuna

ukupna duljina robota u smjeru osi x, y i z u odnosu na kutove zakreta. Od koordinata prihvatnice oduzmu se dobivene vrijednosti i na taj način izračuna ofset robota. Nakon toga možemo početi iscrtavati robota od postolja prema prihvatnici. Vrh robota i realna pozicija prihvatnice tada se poklapaju i gibanje robota je kontinuirano i precizno.

4. ZAKLJUČAK

Kako u dinamičkoj okolini robot i čovjek stalno surađuju potrebno je poznavati položaj i stanje kako čovjeka tako i robota te prepreka u radnom prostoru. Robot informacije o svojoj okolini prikuplja putem senzora i ako su prikupljene informacije dobro protumačene robot može sam odlučiti o svom ponašanju bez da ugrozi čovjeka ili sebe.

Zbog toga je razvijen program koji koristi dva Microsoft Kinect uređaja kako bi se oblikovala virtualna okolina koja odražava podatke prikupljene vizijskim sustavom. Program je implementiran na paralelnu i klasičnu strukturu dok je vizijski sustav robota primijenjen na robotu paralelne strukture zbog jednostavnosti rada sa fiksnom kamerom.

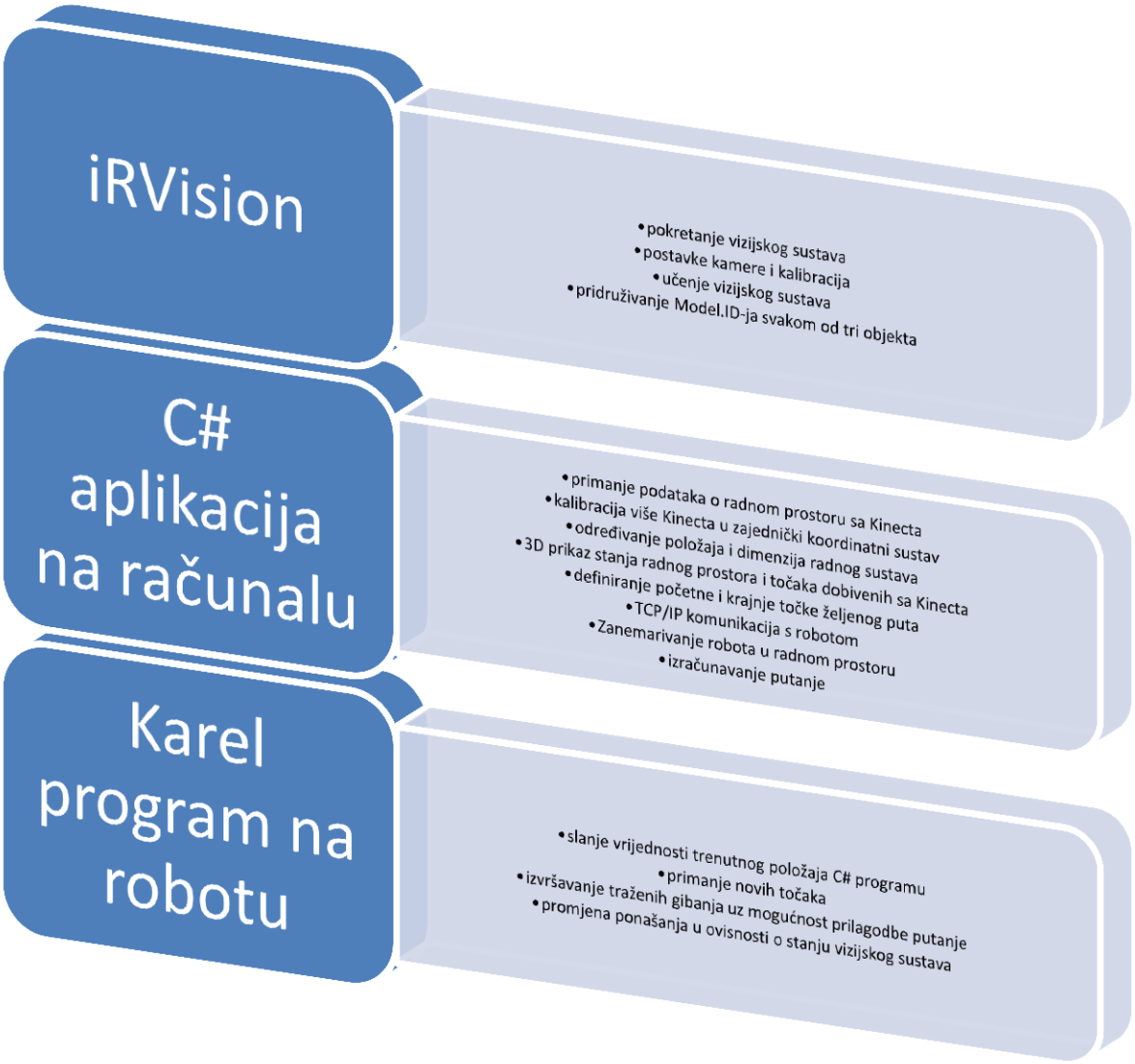
Percepcija robota u radnoj okolini dobiva se putem dva Microsoft Kinect uređaja koje mjere stanje radnog prostora i detektiraju prepreke na temelju čega robot prilagođava svoju putanju. Vizijski sustav na robotu služi za detaljno prepoznavanje oblika i prikupljanje informacija o njima na temelju kojih robot odlučuje o svojoj sljedećoj akciji za vrijeme prije definiranog gibanja. Na robotu je napravljen program koji omogućuje dvosmjernu komunikaciju između robota i računala, a pomoću rastavljene putanje na više dijelova riješen je problem nemogućnosti mijenjanja putanje i odlučivanja tijekom izvođenja programa.

U radu je ispitano ponašanje robota s obzirom na stanje vizijskog sustava, a isto se može napraviti i korištenjem drugih senzora. Tako u budućnosti možemo promatrati ponašanje robota s obzirom na stanje više senzora i koristeći primjerice onotologije i Bayesove mreže robot može samostalno odlučivati o svojoj sljedećoj akciji.

Implementacija na klasičnoj strukturi pokazala se boljom u odnosu na delta strukturu jer je zanemarivanje prostora koji zauzima robot znatno poboljšano, a time i točnije i sigurnije izbjegavanje prepreka. Problem je što je i na jednoj i drugoj strukturi gibanje robota i dalje isprekidano što bi se moglo riješiti prebacivanjem programa na drugi robotski sustav, poput Kukinog LWR-a koji ima ugrađenu mogućnost mijenjanja putanje tijekom izvođenja gibanja preko FRI sučelja.

Također, moguće je spojiti oba programa na način da koristimo vizijski sustav na klasičnoj strukturi koji ima pomičnu kameru. Na taj način robot bi mogao detektirati neki objekt koji mu je ušao u radni prostor, doći do istoga te ga snimiti i na temelju dobivenih podataka odlučiti što dalje.

Pregled napravljenih elemenata sustava i njihove funkcije prikazan je slikom.



Slika 31 Pregled strukture i funkcija izrađenog sustava

LITERATURA

[1] FANUC Robotics SYSTEM R-30iA Controller KAREL Reference Manual, 2007.

FANUC Robotics America, Inc.

[2] Pro C# 5.0 and the .NET 4.5 Framework, Andrew Troelsen, 2012.

[3] Diplomski rad, Tomislav Tomašić, 2012.

[4] Microsoft Visual C# Step by Step, John Sharp, 2010.

[5] Programiranje FANUC robota – osnove, Marko Švaco, Bojan Šekoranja, Bojan Jerbić, Zagreb, 2011.

PRILOG 1 Kod za određivanje ponašanja robota

```
PROGRAM KinectRobot

VAR
 group_no,prog_index,int_value,STATUS,entry,port: INTEGER

 file_var : FILE
 S, reply, message, prog_name : STRING[128]
 camera_view,j, n_bytes, REG, g, counter : INTEGER
 real_flag, grip : BOOLEAN
 config_var:CONFIG

 P1, P2, P3, P4 : XYZWPR
 axisNum, delayCount: INTEGER
 assemble : STRING[128]

 x, y, z : STRING[128]

 diff, real_value : REAL

 ----------VANJSKE RUTINE----------
 ROUTINE OPEN_FILE_(FILE_ : FILE; TAG_ : STRING) FROM LIB_FILE
 ROUTINE CLOSE_FILE_(FILE_ : FILE; TAG_ : STRING) FROM LIB_FILE


 ------------------------------------------------------------
```

```
BEGIN
        $GROUP[1].$UFRAME = $MNUFRAME[1,1];
        $GROUP[1].$UTOOL =  $MNUTOOL[1,1]
        $GROUP[1].$MOTYPE=LINEAR;
        $GROUP[1].$SPEED=300
        $GROUP[1].$TERMTYPE=NODECEL

        port=5555

        SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$OPER',0,STATUS) ;
        SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$STATE',0,STATUS) ; DELAY 20
        SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$COMMENT','SOUND',STATUS) ;
        SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$PROTOCOL','SM',STATUS) ;
         SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$REPERRS','FALSE',STATUS) ;
        SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$TIMEOUT',9999,STATUS) ;
        SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$PWRD_TIMEOUT',0,STATUS) ;
        SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$SERVER_PORT',port,STATUS) ;

        SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$OPER',3,STATUS);
        SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$STATE',3,STATUS) ;

        CLOSE_FILE_(file_var,'S8:')
        OPEN_FILE_(file_var,'S8:')
        DELAY 10 ;
```

```
 CNV_STR_CONF('nut000', config_var, STATUS)


P1=CURPOS(0,0)
P2=CURPOS(0,0)
P3=CURPOS(0,0)
P4=CURPOS(0,0)
grip = FALSE;
n_bytes = 0
delayCount = 0
counter = 0

P4.x=-700
P4.y=-10
P4.z=100

FOR j=1 TO 100 DO
        BYTES_AHEAD (file_var, n_bytes, STATUS)
        IF n_bytes=0 THEN; GOTO connected; ENDIF

        READ file_var (S::1)
        WRITE(S)
ENDFOR

connected::

WRITE(CR)
```

```
WHILE TRUE DO
        BYTES_AHEAD (file_var, n_bytes, STATUS)

        IF n_bytes = 0 THEN
                DELAY 1
        ENDIF

        IF n_bytes > 0 THEN
                READ file_var (message:: 1) --Determine Command

                IF message = 'A' THEN
                        READ file_var (message::1)
                        assemble = ''
                        WHILE message <> ';' DO
                                assemble = assemble + message
                                READ file_var (message::1)
                        ENDWHILE
                        CNV_STR_REAL(assemble, P1.x)

                        READ file_var (message::1)
                        assemble = ''
                        WHILE message <> ';' DO
                                assemble = assemble + message
                                READ file_var (message::1)
                        ENDWHILE
```

```
                CNV_STR_REAL(assemble, P1.y)

                READ file_var (message::1)
                assemble = ''
                WHILE message <> ';' DO
                        assemble = assemble + message
                        READ file_var (message::1)
                ENDWHILE
                CNV_STR_REAL(assemble, P1.z)

                READ file_var (message::1)
                assemble = ''
                WHILE message <> ';' DO
                        assemble = assemble + message
                        READ file_var (message::1)
                ENDWHILE
                CNV_STR_REAL(assemble, $GROUP[1].$SPEED)

        ENDIF

        IF message = 'S' THEN
                --CANCEL
                P1 = P3
                --P3 = CURPOS(0,0)
        ENDIF
ENDIF
```

```
P2 = CURPOS(0,0)


diff = $GROUP[1].$SPEED /10;

IF (  (P2.x - P3.x)*(P2.x - P3.x) + (P2.y - P3.y)*(P2.y - P3.y)  + (P2.z - P3.z
)*(P2.z - P3.z) ) < diff*10 THEN


        IF(P2.x - P1.x < 0) THEN

                P3.x = P2.x + diff;

        ENDIF

        IF(P2.y - P1.y < 0) THEN

                P3.y = P2.y + diff;

        ENDIF

        IF(P2.z - P1.z < 0) THEN

                P3.z = P2.z + diff;

        ENDIF


        IF(P2.x - P1.x > 0) THEN

                P3.x = P2.x - diff;

        ENDIF

        IF(P2.y - P1.y > 0) THEN

                P3.y = P2.y - diff;

        ENDIF

        IF(P2.z - P1.z > 0) THEN

                P3.z = P2.z - diff;

        ENDIF


        IF (ABS(P2.x - P1.x) < diff) THEN
```

```
            P3.x = P1.x;
        ENDIF
        IF (ABS(P2.y - P1.y) < diff) THEN
            P3.y = P1.y;
        ENDIF
        IF (ABS(P2.z - P1.z) < diff) THEN
            P3.z = P1.z;
        ENDIF

        MOVE TO P3 NOWAIT
        WRITE file_var ('M', CR)

ELSE
        counter = counter + 1

        IF (counter = 100) THEN
            counter = 0
            start::
            SET_INT_REG(1, 0, STATUS)
            CALL_PROG ('SHAPES',prog_index)
            GET_REG (1,real_flag,int_value,real_value,STATUS)

            IF (int_value)=1 THEN
                DELAY 1000
                GOTO start
            ENDIF
```

```
                    IF (int_value)=2  THEN
                        MOVE TO P4 NOWAIT
                    ENDIF

                    IF (int_value)=3 THEN
                        CALL_PROG('VPMIA',prog_index)
                    ENDIF

                    IF (int_value)<>0 THEN
                        P3 = CURPOS(0,0)
                        GOTO start
                    ENDIF
                ENDIF
            ENDIF

            ----------Send Axis status to PC
            CNV_REAL_STR(P2.x,2,2, x)
            CNV_REAL_STR(P2.y,2,2, y)
            CNV_REAL_STR(P2.z,2,2, z)
            WRITE file_var ('A[0]=' +x + ';A[1]=' + y + ';A[2]=' + z + ';', CR)

        ENDWHILE

        CLOSE_FILE_(file_var,'S8:')
END KinectRobot
```

PRILOG 2 Kod za dobivanje kutova i pozicije prihvatnice robota

```
reg_no=1
jpos=CURJPOS(0,0)
SET_JPOS_REG(reg_no,jpos,STATUS)
cur_pos=GET_JPOS_REG(1,STATUS)
CNV_JPOS_REL(cur_pos,jp_org,STATUS)

----------Send Axis and Joint position status to PC
CNV_REAL_STR (P2.x,2,2, x)
CNV_REAL_STR (P2.y,2,2, y)
CNV_REAL_STR (P2.z,2,2, z)
WRITE file_var ('A[0]=' +x + ';A[1]=' + y + ';A[2]=' + z + ';', CR)

CNV_REAL_STR(jp_org[1],2,2, q1)
CNV_REAL_STR(jp_org[2],2,2, q2)
CNV_REAL_STR(jp_org[3],2,2, q3)
CNV_REAL_STR(jp_org[5],2,2, q4)
WRITE file_var ('Q[0]=' +q1 + ';Q[1]=' + q2 + ';Q[2]=' + q3 + ';Q[3]=' + q4+ ';', CR)
```

PRILOG 3 Kod za crtanje robota u OpenGL sučelju

```
  void FindSquaresWithRobotNewStructure()
        {
            int a = (int)(_robotViewModel.Axes[0].Value / workSpace.elementSize + 20f);
            int c = (int)(_robotViewModel.Axes[1].Value / workSpace.elementSize + 5);
            int b = (int)(_robotViewModel.Axes[2].Value / workSpace.elementSize - 5f);

            a = workSpace.width / workSpace.elementSize - a;
            c = workSpace.depth / workSpace.elementSize - c;

            if (a < 0)
                a = 0;
            else if (a > workSpace.width / workSpace.elementSize - 1)
                a = workSpace.width / workSpace.elementSize - 1;

            if (b < 0)
                b = 0;
            else if (b > workSpace.height / workSpace.elementSize - 1)
                b = workSpace.height / workSpace.elementSize - 1;

            if (c < 0)
                c = 0;
            else if (c > workSpace.depth / workSpace.elementSize - 1)
                c = workSpace.depth / workSpace.elementSize - 1;


            //workSpace.elements[a, b, c].hasRobot = true;

            robotAngle0 = (_robotViewModel.Angles[0].Value+90) / 180.0 * 3.14;
            robotAngle1 = (90 - _robotViewModel.Angles[1].Value) / 180.0 * 3.14;
            robotAngle2 = (_robotViewModel.Angles[2].Value) / 180.0 * 3.14;
            robotAngle3 = ((-_robotViewModel.Angles[3].Value - 180)-RobotAngle2) /
180.0 * 3.14;


            int L1 = 650;
            int L2 = 550;
            int L3 = 350;

            double x = 0;
            double y = 0;
            double z = 0;

            double x1 = 0;
            double y1 = 0;
            double z1 = 0;

            double x2 = 0;
            double y2 = 0;
            double z2 = 0;


            int xSquare = 0;
            int ySquare = 0;
```

```
            int zSquare = 0;

            double cosA0 = Math.Cos(robotAngle0);
            double sinA0 = Math.Sin(robotAngle0);

            double cosA1 = Math.Cos(robotAngle1);
            double sinA1 = Math.Sin(robotAngle1);

            double cosA12 = Math.Cos(robotAngle2);
            double sinA12 = Math.Sin(robotAngle2);

            double cosA23 = Math.Cos(robotAngle3);
            double sinA23 = Math.Sin(robotAngle3);

            x = cosA1 * cosA0 * L1;
            y = sinA1 * L1;
            z = cosA1 * Math.Abs(sinA0) * L1;

            x1 = cosA12 * cosA0 * L2 + x;
            y1 = sinA12 * L2 + y;
            z1 = cosA12 * Math.Abs(sinA0) * L2 + z;

            x2 = cosA23 * cosA0 * L3 + x1;
            y2 = sinA23 * L3 + y1;
            z2 = cosA23 * Math.Abs(sinA0) * L3 + z1;


            Vector3 _robotPositionOffset1;

            _robotPositionOffset1.X = a * workSpace.elementSize - (float)(x2) + 2 *
workSpace.elementSize;
            _robotPositionOffset1.Y = b * workSpace.elementSize - (float)(y2) +
workSpace.elementSize;
            _robotPositionOffset1.Z = c * workSpace.elementSize - (float)(z2) + 2 *
workSpace.elementSize;

            Vector3 start = new Vector3(_robotPositionOffset1.X,
_robotPositionOffset1.Y, _robotPositionOffset1.Z);
            offset.Add(start);

            #region od_offseta


            for (int i = 0; i < L1; i += 10)
            {
                x = cosA1 * cosA0 * i + _robotPositionOffset1.X;
                y = sinA1 * i + _robotPositionOffset1.Y;
                z = cosA1 * sinA0 * i + _robotPositionOffset1.Z;

                xSquare = (int)(x / workSpace.elementSize);
                ySquare = (int)(y / workSpace.elementSize);
                zSquare = (int)(z / workSpace.elementSize);

                PlaceRobotAround(xSquare, ySquare, zSquare, 2);
            }

            for (int i = 0; i < L2; i += 10)
```

```
            {
                x1 = cosA12 * cosA0 * i + x;
                y1 = sinA12 * i + y;
                z1 = cosA12 * sinA0 * i + z;

                xSquare = (int)(x1 / workSpace.elementSize);
                ySquare = (int)(y1 / workSpace.elementSize);
                zSquare = (int)(z1 / workSpace.elementSize);

                PlaceRobotAround(xSquare, ySquare, zSquare, 1);
            }

            for (int i = 0; i < L3; i += 10)
            {

                x2 = cosA23 * cosA0 * i + x1;
                y2 = sinA23 * i + y1;
                z2 = -cosA23 * sinA0 * i + z1;


                xSquare = (int)(x2 / workSpace.elementSize);
                ySquare = (int)(y2 / workSpace.elementSize);
                zSquare = (int)(z2 / workSpace.elementSize);

                PlaceRobotAround(xSquare, ySquare, zSquare, 1);
            }
            #endregion od_offseta

            //Tool position
            workSpace.start = new Vector3(a,b,c);
        }

        public void PlaceRobotAround(int xSquare, int ySquare, int zSquare, int radius)
        {
            for (int i = xSquare - radius; i < xSquare + radius; i++)
            {
                for (int j = zSquare - radius; j < zSquare + radius; j++)
                {
                    if (!(i < 0 || i > workSpace.width / workSpace.elementSize - 1 ||
                            ySquare < 0 || ySquare > workSpace.height /
workSpace.elementSize - 1 ||
                            j < 0 || j > workSpace.depth / workSpace.elementSize - 1))
                    {
                        workSpace.elements[i, ySquare, j].hasRobot = true;
                    }
                }
            }

        }
```